Sebastian Eube

Programm zur Auswertung von NMR-Experimenten am RNA-Stem

BACHELORARBEIT

HOCHSCHULE MITTWEIDA (FH)

UNIVERSITY OF APPLIED SCIENCES

Fachbereich Mathematik/Physik/Informatik

Mittweida, September 2009

Sebastian Eube

Programm zur Auswertung von NMR-Experimenten am RNA-Stem

eingereicht als

BACHELORARBEIT

an der

HOCHSCHULE MITTWEIDA (FH)

UNIVERSITY OF APPLIED SCIENCES

Fachbereich Mathematik/Physik/Informatik

Mittweida, September 2009

Erstprüfer: Prof. Dr. G. Werner
Zweitprüfer: Prof. Dr. D. Labudde

Vorgelegte Arbeit wurde verteidigt am: n.v. 2009

Bibliographische Beschreibung:

Sebastian Eube:
Programm zur Auswertung von NMR-Experimenten am RNA-Stem 2009.
- 37 S. Mittweida,
Hochschule Mittweida (FH) - University of Applied Sciences,
Fachbereich Mathematik/Physik/Informatik, Bachelorarbeit, 2009

Kurzreferat:

Ziel der Bachelorarbeit ist es, ein Programm zur Auswertung von bestimmten Genetikexperimenten zu entwickeln. Durch die Automatisierung der Auswertung solcher Experimente wird die Forschung vorangetrieben und die Chance auf Fehler wird dezimiert. Um die Arbeit für alle verständlich zu machen, werden zu Beginn die biologischen Hintergründe, die der Arbeit zu Grunde liegen, erläutert, um danach auf die aus den Experimenten gewonnenen Dateien einzugehen. Anschließend wird das Konzept sowie die Realisierung näher betrachtet. Zum Schluß erfolgt eine Zusammenfassung und ein Ausblick auf mögliche Programmerweiterungen.

Danksagung

In erster Linie gilt mein besonderer Dank Prof. Dr. rer. nat. Dirk Labudde, der mich zu dem Thema inspiriert hat und jederzeit für Fragen oder Kritiken zur Verfügung stand. Er hat diese Arbeit ermöglicht, in dem er mir nicht nur die Angst vor einem biologischen Thema genommen hat, sondern auch die Geduld aufgebracht hat, mit mir jedes Thema intensiv durchzugehen. Bedanken möchte ich mich auch bei Prof. Dr. Mario Schubert, dem Urheber der Aufgabe, für seine Hinweise und Anmerkungen, sowie Thomas Aeschebacher, der mir alle wichtigen Ergebnisse seiner Experimente an der ETH Zurüch zukommen ließ. Weiterhin gilt mein Dank Prof. Dr. rer. nat. Günter Werner, der als Betreuer der Hochschule auch jederzeit für Fragen zur Verfügung stand. Letztendlich möchte ich noch meiner Familie danken, die mich auch in arbeitsreichen Zeiten immer motiviert hat.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

δ	chemische Verschiebung in ppm
ρ	Korrelationskoeffizient
μ	Mittelwert
σ	Standardabweichung
A	Adenin, Baustein der Ribonukleinsäure
BRMB	Biological Resonance Magnetic Data Bank
C	Cytosin, Baustein der Ribonukleinsäure
DNA	Desoxyribonukleinsäure (engl.: deoxyribonucleic acid)
FID	engl.: Free induction decay
G	Guanin, Baustein der Ribonukleinsäure
H	Wasserstoff
HSQC	Heteronukleare Einzelquantumkohärenz (engl.: heteronuclear single quantum coherence)
INEPT	engl.: Insesitive Nuclei Enhanced by polarization Transfer
N	Stickstoff
NMR	Kern(spin)resonanz(spektroskopie) (engl.: nuclear magnetic resonance)
NOESY	Kern-Overhauser-Effekt-Spektroskopie (engl.:nuclear overhauser enhancement spectroscopy)
PDB	Protein Data Bank
RNA	Ribonukleinsäure (engl.: ribonucleic acid)
TOCSY	vollständige Korrelationsspektroskopie (engl.: total correlated spectroscopy)
U	Uracil, Baustein der Ribonukleinsäure

Kapitel 1

Einleitung

1.1 Aufgabe

In der heutigen Zeit werden immer mehr Dienste für den Computer angeboten, die man früher meist per Hand oder anderen, weniger schnellen, Hilfsmitteln bearbeiten musste.

Besonders in der Forschung sind solche Hilfsmittel (Tools) sehr nützlich, da sie viel Zeit einsparen können. Die Arbeit beschäftigt sich mit einem biologischen Thema, der schnelleren Auswertung von NMR-Experimenten am RNA-Stem, was vorher mühsam per Hand berechnet werden musste und mittels des Programms RASP nun zügiger voran gehen kann. Der Name kommt aus dem englischen und bedeutet soviel wie Vorhersage für die Zuordnung der RNA-Sequenz (**R**N**A** - **S**equenz assignment **P**rediction).

Die Geschichte der RNA begann in den 50er Jahren, als Francis Crick (1916-2004, Gentechniker) sein zentrales Dogma der Molekularbiologie postitulierte. Er erkannte die Schlüsselfunktion der RNA bei der Übertragung genetischer Informationen der DNA zum Protein (Abb. 1.1). Um die RNA entschlüsseln zu können, werden sogenannte NMR-Experimente durchgeführt, die den Aufbau der Säuren zeigen.
Diese Experimente werden unter anderem an der ETH Zürich realisiert und die Ergebnisse können durch das Programm ausgewertet werden. Zusätzlich soll es auch dazu dienen, die vorhandenen Informationen zu visualisieren, um ein besseres Verständnis für Zusammenhänge bei dem Nutzer zu schaffen. RASP basiert auf Java, wobei es sich um eine plattformunabhängige Sprache handelt, um es allen potenziellen Interessenten zugänglich zu machen.

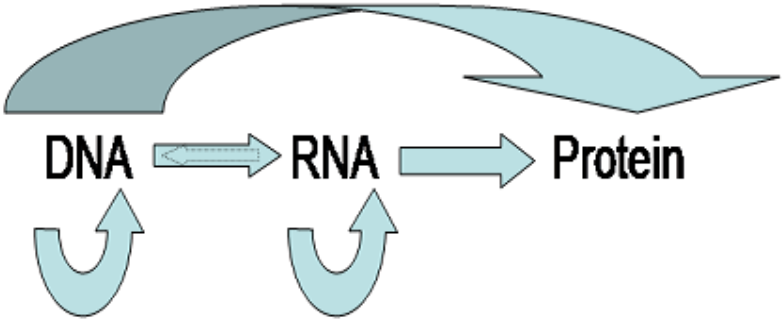


Abbildung 1.1: Von der DNA zum Protein

1.2 Motivation

Die Idee, ein solches Projekt durchzuführen, kam von der ETH Zürich, an der auch die entsprechenden Experimente realisiert werden.
Da die Forschung ein faszinierendes Gebiet für fast alle Menschen darstellt, weckt eine solche Aufgabe schnell das Interesse, sich einmal näher mit dem Thema zu beschäftigen. Vor allem die Entstehung von neuen Organismen stellte bereits in der Schule ein spannendes Fachgebiet dar. Der RNA wird eine besondere Schlüsselfunktion bei der Entstehung neuer Zellen in unserem Körper zugeschrieben, wodurch es nicht schwer fällt, ein Angebot, ein solches Thema zu bearbeiten, anzunehmen.

1.3 Kapitelübersicht

Kapitel 2

Dieses Kapitel liefert das Grundlagenwissen zum weiteren Verständnis der Arbeit. Es beschäftigt sich mit dem Aufbau und der Darstellung der RNA sowie den zugrunde liegenden Experimenten.

Kapitel 3

Hier wird beschrieben, welche Dateien das fertige Programm auswerten sollte und wie diese aufgebaut sind. Es wird außerdem die Quelle der Datensätze beschrieben.

Kapitel 4

Kapitel drei gibt stichpunktartig den Funktionsumfang sowie die Designanforderungen an RASP wieder. Es wird das Konzept für das Programm beschrieben, wobei Werkzeuge aus der Softwaretechnik verwendet wurden.

Kapitel 5

Die Realisierung stellt den wichtigsten Themenkomplex dar, in dem beschrieben wurde, was und vor allem wie bei der Arbeit vorgegangen wurde. Es behandelt auch den mathematischen Ansatz für die Berechnung der Wahrscheinlichkeiten und gibt am Ende noch ein kurzes Beispiel zu der Arbeit mit dem Programm.

Kapitel 6

Letztendlich kommt es noch zu einem Fazit und einem Ausblick auf weitere Arbeiten in dem Gebiet.

Kapitel 2

Theoretische Grundlagen

2.1 Bedeutung und Aufbau der RNA

In diesem Kapitel wird auf den biologischen Hintergrund der Arbeit eingegangen.
Es existieren verschiedene RNA-Typen, die aber alle eine zentrale Rolle bei der Speicherung, Übertragung und Expression genetischer Informationen, sowie bei der Katalyse chemischer Reaktionen in der Zelle spielen. Sehr wesentliche Erkenntnisse lieferte die Forschung der 50er und 60er Jahre des 20. Jahrhunderts. Bei vielen Lebewesen ist die genetische Information in Form von DNA gespeichert. Die Herstellung von Proteinen, den Grundbausteinen der Zelle, aus dieser Information nennt man Genexpression oder Proteinbiosynthese. Dafür muss die DNA zunächst in RNA umgeschrieben werden, dies wird auch als Transkription bezeichnet. Das dabei entstehende Transkript, eine modifizierte Arbeitskopie der DNA, heißt mRNA (messangerRNA).
Die mRNA dient bei der Translation als Vorlage der Synthese von Proteinen, die bei Eukaryonten posttranslational modifiziert werden.

> *Unter Translation versteht man in der Biologie den Vorgang, bei dem in der lebenden Zelle aus einer in einer Abfolge von Nukleotiden codierten Information ein Protein hergestellt wird. Die Translation ist somit der letzte Schritt in einem Prozess, bei dem anhand des genetischen Codes aus Erbinformation ein Eiweißmolekül hergestellt wird (siehe Eiweißsynthese). Dieser letzte Schritt geschieht in lebenden Zellen an besonderen Organellen, den Ribosomen.*
> Bruce Alberts

An diesem Schritt sind zusätzlich auch noch Ribosomen und tRNA (transferRNA) beteiligt. Ribosomen kommen im Zellplasma aller Lebewesen vor und bestehen aus rRNA (ribosomalerRNA) und Proteinen.

Zum besseren Verständnis der Arbeit sollte auch noch erwähnt werden, dass es sich bei der Ribonukleinsäure um Makromoleküle handelt, die aus einer Kette von sogenannten Nukleotiden bestehen. Jedes dieser Nukleotide besteht wiederum aus einer Phosphatgruppe, einer Pentose und einer Nukleinbase. Der aus Pentose und Nukleinbase bestehende Molekülteil wird zusammenfassend auch als Nukleosid bezeichnet (Abb.2.1).

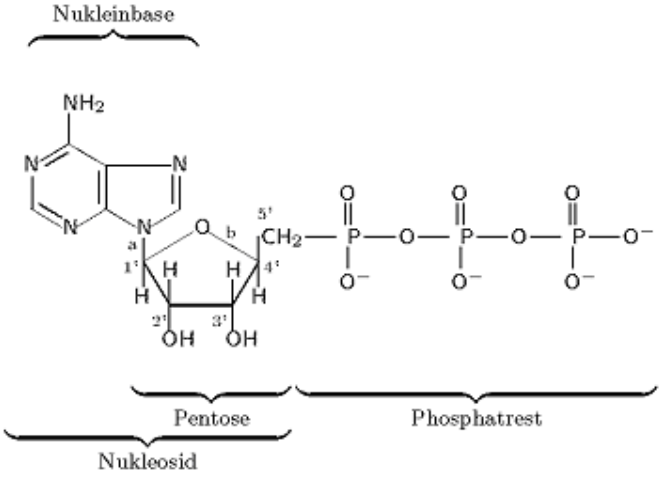


Abbildung 2.1: Teile des Makromoleküls RNA

Die Phosphatgruppen verbinden die einzelnen Nukleotide miteinander und spielen zum weiteren Verständnis dieser Arbeit keine große Rolle.
Wesentlich wichtiger hingegen ist das Nukleosid, welches bei der RNA in die Gruppen Adenosin, Guanosin, Cytidin und Uridin unterteilt wird. Diese Namen entspringen den RNA-Nukleinbasen Adenin, Guanin, Cytosin und Uracil und stellen den einzigen Bereich dar, in dem sich die verschiedenen Nukleoside unterscheiden. Chemisch werden diese in Purin-und Pyrimidinbasen untergliedert. Zu den Purinbasen zählen die natürlich vorkommenden Basen Guanin und Adenin. Uracil und Cytosin sind Pyrimidinbasen.

2.2 Darstellung der RNA

Bei der RNA unterscheidet man drei verschiedene Darstellungsarten: der Primär-, Sekundär- und Tertiärstruktur. Als Primärstruktur wird die einfache Abfolge der Nukleotide (Nukleotidsequenz) bezeichnet (Abb. 2.2). Dabei wird der Beginn der Sequenz als 5'-Ende und das Ende als 3'-Ende bezeichnet. Der Anfang steht für die Phosphatgruppe am C5' der Ribose. Diese Unterscheidung ist wichtig, da die Sequenz nicht symmetrisch ist.

GGGUCAUCAGGACGAUGACCC

Abbildung 2.2: Primärstruktur einer RNA-Sequenz

Die RNA liegt im Gegensatz zur DNA meist nur als Einzelstrang vor. Doch auch diese können sich miteinander paaren und gehen dabei Wasserstoffbrückenbindungen ein.
Diese Form wird als Stem bezeichnet, wofür zunächst auch RASP entwickelt wurde. Falls ein AU-Basenpaar an einem der beiden Wasserstoffatome am N_6 am Adenin mit dem Sauerstoffatom des Uracil sowie das H_3 des Uracil mit dem N_1 des Adenin Wasserstoffbrückenbindungen eingehen, spricht man auch von einem Watson-Crick-Basenpaar oder Watson-Crick-Verbindungen. Dies stellt nur ein Beispiel dar, wann ein solches Basenpaar als Watson-Crick-Paar bezeichnet wird. Die Darstellung dieser Paarungen wird als Sekundärstruktur bezeichnet (Abb. 2.3). Dabei wird der Teil, in dem Wasserstoffbrückenbindungen zu finden sind, als Stem und der Teil ohne Verbindungen als Loop bezeichnet.

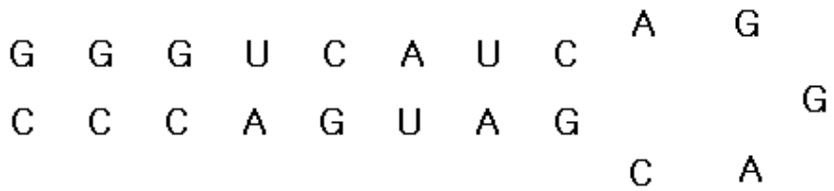


Abbildung 2.3: Sekundärstruktur einer RNA-Sequenz

Sobald das Molekül dreidimensional dargestellt wird, handelt es sich bei der RNA um die Tertiärstruktur (Abb. 2.4). Dabei interessieren vor allem die Bindungswinkel sowie der räumliche Abstand einzelner Atome zueinander.

Bestimmt werden können diese beispielsweise durch NMR- oder Röntgenstrukturanalysen.

Abbildung 2.4: Tertiärstruktur einer RNA-Sequenz (Vossmann: Urheberrecht: http://www.wwpdb.org/)

2.3 NMR-Experiment

Den Kernspinresonanzspektroskopie-Experimenten kommt ein großer Stellenwert in der Aufklärung von RNA-Strukturen bei.
Damit können beispielsweise molekulare Erkennungsprozesse, intramolekulare Dynamik und Proteinentfaltung untersucht werden.

Um das Experiment besser verstehen zu können, wird es nachfolgend detailliert erläutert. Ein Atomkern besteht aus Protonen und Neutronen. Die Ordnungszahl, welche für die Anzahl der Protonen im Kern steht, ist Z.

I steht für die Kernspin-Quantenzahl und kann Werte von $I = 0, \frac{1}{2}, 1, 1\frac{1}{2}, \ldots, 6$ annehmen.

Die Nukleonenzahl M gibt die Summe aller Protonen und Neutronen an. Ist diese Zahl ungerade, so ist der Kernspin I halb- ansonsten ganzzahlig. Kerne mit ungerader Protonen und gerader Neutronenzahl haben niemals einen Kernspin von Null. Der gequantelte Spindrehimpuls $\vec{J}$ wird über den Kernspin definiert.

$$|\vec{J}| = \sqrt{I(I+1)}\hbar$$

$\vec{J}$ ... Spindrehimpuls

I ... Kernspin-Quantenzahl

$\hbar$ ... reduziertes Plancksches Wirkungsquantum $\hbar = \frac{h}{2\pi}$

Das magnetische Moment μ eines Atomkerns ist zu $\vec{J}$ direkt proportional:

$$\vec{\mu} = \gamma\vec{J} \Rightarrow |\vec{\mu}| = \gamma\hbar\sqrt{I(I+1)}$$

Isotope mit I $\neq$ 0 besitzen demnach ein magnetisches Moment. Der Proportionalitätsfaktor ist das gyromagnetische Verhältnis. Wird ein Atomkern, der einen Spindrehimpuls besitzt, in ein externes Magnetfeld B_0 gebracht, so richtet sich der Vektor $\vec{J}$ parallel zum Magnetfeld oder genau entgegengesetzt aus. Er hält jedoch einen von 2I+1 möglichen Winkeln α_i ein, unter welchem er die Achse des Magnetfeldes in einer
Präzisionsbewegung umkreist. Die Größe der Winkel beträgt $\alpha_i = arccos\sqrt{\frac{m}{m+1}}$, wobei die magnetische Quantenzahl m Werte mit ganzzahligen Differenzen untereinander zwischen -I und +I annehmen kann.

Die meisten der für die NMR-Spektroskopie interessanten Isotope haben einen Kernspin von $I = \frac{1}{2}$. So können die Kerne zwei verschiedene Zustände einnehmen:

m kann den Wert +$\frac{1}{2}$ besitzen, man spricht von einer parallelen Ausrichtung im Magnetfeld (↑), oder den Wert -$\frac{1}{2}$ bei einer antiparallelen Ausrichtung (↓). Der Zustand bei Ausrichtung mit dem Magnetfeld ist energieärmer und tritt deshalb etwas häufiger auf. Das magnetische Moment des Kerns umkreist die Achse des Magnetfeldes mit der Larmor-Frequenz ω_0. Sie entspricht der im Bereich der Radiowellen liegenden Resonanzfrequenz des Kerns und ist abhängig vom gyromagnetischen Verhältnis sowie von der Stärke des anliegenden Magnetfeldes.

Nach der Regel von Lenz bewirkt das Anschalten eines Magnetfeldes die Induktion eines Stromes.

In aromatischen Systemen entsteht als Anisotropieeffekt ein durch die π-Elektronen in den ρ_z-Orbitalen verursachter Ringstrom in der Ringebene, der wiederum ein Magnetfeld erzeugt. Dieses Magnetfeld ist im Inneren des Rings dem äußeren Magnetfeld entgegengerichtet. In der Nähe der Protonen ist es jedoch mit dem Magnetfeld B_0 ausgerichtet und verstärkt dieses. Mit dem Anlegen eines zweiten, zum ersten senkrechten Magnetfeldes, finden im NMR-Spektrometer Übergänge zwischen den verschiedenen
Energieniveaus statt, wenn die Resonanzbedingung erfüllt ist (die Frequenz der Strahlung, die dem Energieunterschied der Energieniveaus entspricht). Die Kernspins werden in eine andere Richtung gedreht.

Dadurch werden einige der energieärmeren Kerne durch die elektromagnetische Strahlung auf ein höheres Energieniveau gebracht.

Die Stärke des Ringstromeffekts hängt von der Orientierung des Rings im Magnetfeld ab. Er wird maximal, wenn die Fläche des Benzolrings(C_6H_6) senkrecht zum B_0-Feld ausgerichtet ist. Bei der gepulsten Fourier-Transformation-NMR wird die Resonanzbedingung dadurch erfüllt, dass ein nur einige μs andauernder Puls an Radiostrahlung mit einer bestimmten Frequenz ausgesendet wird. Ein so kurzer Impuls erzeugt ein symmetrisches Frequenzband rund um die Anregungsfrequenz. Je kürzer der Impuls, desto breiter wird dieses Frequenzband. Damit wird die Resonanzfrequenz aller Kerne eines Atomtyps gleichzeitig getroffen.

Alle angeregten Kernspins geben anschließend die aufgenommene Energie als Radiowellen wieder ab. Ein aus vielen Überlagerungen bestehendes Signal wird aufgenommen. Die Fourier-Transformation überführt die Daten aus der Zeit- in die Frequenzdomäne. Die einen Kern umgebenden Elektronen erzeugen ein Magnetfeld, das den Kern geringfügig vom Hauptfeld abschirmt. Die Ringprotonen sind nur geringfügig abgeschirmt, die chemische Verschiebung ist deswegen groß.

Aufgrund der chemischen Umgebung unterscheiden sich die Larmor-Frequenzen der ein-

zelnen Kerne. Jedes Peaksignal kann somit einem angeregten Kern im Molekül zugeordnet werden. Er gibt Auskunft über die chemische Umgebung des Kerns. Dieser Effekt wird als chemische Verschiebung δ bezeichnet und ist wie folgt definiert:

$$\delta = \frac{\omega_{Signal} - \omega_{Referenz}}{\omega_{Referenz}} * 10^6 ppm$$

Hierbei ist die Standardfrequenz $\omega_{Referenz}$ das Signal der Methylgruppe von Tetramethylsilan. Diese Substanz wird aus praktischen Gründen als Referenz verwendet, denn sie liefert im Gegensatz zu vielen anderen Substanzen nur ein einziges Signal, welches sehr intensiv ist.

Eine Form der NMR-Experimente ist das TOCSY-Experiment (total correlated spectroscopy, vollständige Korrelationsspektroskopie). Bei diesem Experiment wird die Impulsfolge 90° - t_1 - Spin-Lock - FID verwendet. Dabei spielt der Spin-Lock eine besonders große Rolle, da die „Reichweite“ der nachweisbaren Korrelationen ganz entscheidend von seiner Dauer abhängt. Spin-Lock bedeutet, dass sich während seiner Dauer die Kerne nur im schwachen Hochfrequenzfeld B_1 des Spin-Locks befinden. Deswegen werden die chemischen Verschiebungsdifferenzen der Protonen sehr klein und die skalaren Kopplungen überwiegen, was dazu führt, dass sich die Spinzustände mischen und Magnetisierung übertragen werden kann. Dies bedeutet also, dass, sobald ein Proton durch einen selektiven Impuls angeregt wird, die Magnetisierung auf alle anderen Protonen des Spinsystems übertragen wird, nicht nur auf die mit einer direkten skalaren Kopplung zum angeregten Proton. Mit Hilfe dieses Impulses werden vor allem die ^{5}H und ^{6}H-Atome beschossen, wodurch auch nur diese gemessen werden können. Es handelt sich dabei also um ein relativ schnelles, aber dafür ungenaues Experiment.

Eine weitere Form der NMR-Experimente ist das HSQC-Experiment. Die Strategie dieses Experimentes besteht darin, dass zunächst eine ^{1}H-Magnetisierung M_H durch eine normale INEPT-Impulsfolge auf die ^{13}C-Magnetisierung M_C unter Verstärkung übertragen wird. Bei einem INEPT-Experiment wird dabei zwischen Impulsen, die nur auf Protonen und jenen, die nur auf ^{13}C-Kerne wirken, unterschieden. Der Vorteil ist, dass die Signalverstärkung im Gegensatz zu anderen Verfahren durch unselektive Impulse erreicht wird. Außerdem werden die Signale aller ^{13}C-Kerne eines Moleküls, die gleiche oder ähnliche skalare Kopplungskonstanten zu einem Nachbarproton aufweisen, verstärkt. Nachdem die Magnetisierung übertragen wurde, sollen sich im zweiten Schritt die Magnetisierungsvektoren M_C in der (inkrementierten) Zeit t_1 entwickeln, worauf im letzten Schritt diese
Magnetisierung (Kohärenz) durch ein inverses INEPT-Experiment auf die Protonen zurück transferiert wird. Abschließend werden noch ^{1}H-Resonanzen detektiert.

Zu guter Letzt wurde RASP noch für ein drittes Experiment geschrieben, welches sich kurz NOESY nennt. Dabei erfolgt der Magnetisierungstransfer nicht wie bisher zwischen zwei skalar gekoppelten Kernen, sondern durch Dipol-Dipol-Wechselwirkungen durch den Raum. Dieser Effekt wird auch Kern-Overhauser-Effekt genannt. Zu Beginn liegen bei diesem Experiment die makroskopischen Magnetisierungsvektoren in z-Richtung, die auch der Richtung des angelegten Feldes entsprechen. Ein $90°_X$-Impuls dreht diese Vektoren in die Richtung der y-Achse und während der Zeit t_1, die wie bei den vorangegangenen Experimenten inkrementiert wird, rotieren die Magnetisierungsvektoren um die z-Achse und fächern dabei wegen ihrer unterschiedlichen Lamorfrequenzen auf. Es folgt ein zweiter $90°_X$-Impuls, der die vorhandenen y-Komponenten der beiden Magnetisierungsvektoren solange dreht, bis sie in Richtung der z-Achse liegen.
In dem folgenden Zeitintervall relaxiert das Spinsystem und die Magnetisierung wird zwischen den Magnetisierungsvektoren übertragen. Dieser Prozess ist auch als „cross-relaxation“ oder „cross-polarization“ bekannt. Ein letzter $90°_X$-Impuls dreht nun diese polarisierten Magnetisierungsvektoren in die y-Richtung, so dass jetzt der FID detektiert werden kann. Eine Fourier Transformation bezüglich der Zeiten ergibt nun das zweidimensionale Spektrum, welches als Peak-Liste dem Programm zur Verfügung gestellt werden kann.

Kapitel 3

Voraussetzung

Um die Ergebnisse der NMR-Experimente auszuwerten, benötigt man ein festes Dateiformat. RASP arbeitet mit drei verschiedenen Arten von Dateien im Textformat.

3.1 Fasta-File

Die erste Art wird als Fasta-File bezeichnet und beinhaltet die Informationen zur Sequenz. Diese beginnt immer mit einem Größer-als-Zeichen $(>)$ mit der darauf folgenden Kopfzeile, in welcher die Bezeichnung der Sequenz angegeben wird. Nun können weitere Informationen folgen, die mittels '—' oder ';' getrennt werden. In Abbildung 3.1 wird zusätzlich noch angegeben, woher die Sequenz kommt und dass es sich um eine Verkettung von Nukleinbasen handelt. Es folgt die Bezeichnung 'SEQUENCE', die den Beginn der Sequenz einleitet und direkt daran angegliedert die Auflistung der Nukleinbasen von 5'-Anfang bis 3'-Ende. Es existiert zwar keine Standard-Dateierweiterung für diese Art von Dateien, aber viele verwenden 'fasta' als Erweiterung.

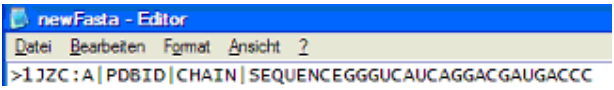


Abbildung 3.1: Fasta-Datei

3.2 Peak-liste

Eine weitere, sehr wichtige, Dateiart ist die sogenannte Peakliste (Abb. 3.2), nicht zu verwechseln mit der peak-Datei von Musikprogrammen.

Für die Speicherung der Frequenzen, die bei den NMR-Experimenten gemessen werden, wird eine solche Datei benötigt. Diese Datei besteht genau wie das Fasta-File zunächst aus einer Kopfzeile. Diese beginnt häufig mit dem Wort 'Assignment' und den beiden Wörtern 'w1' und 'w2'. Unter dieser Kopfzeile werden nun die Ergebnisse der Frequenzmessungen sowie zusätzliche Informationen geschrieben.

Dabei werden unter Assignment die Nukleinbasen mit ihren Atomen aufgelistet, die zu der ermittelten Frequenz gehören. Sind diese noch nicht bekannt, wird der Wert '?-?' benutzt. Die Werte unter 'w1' und 'w2' stehen für die Freuenzgrößen.

Eine spezielle Dateierweiterung für diese Art von Dateien existiert nicht, es wird aber oft das Wort 'assign' oder 'noassign' am Ende angehangen um zu verdeutlichen, ob die Informationen in der Datei bereits ausgewertet wurden.

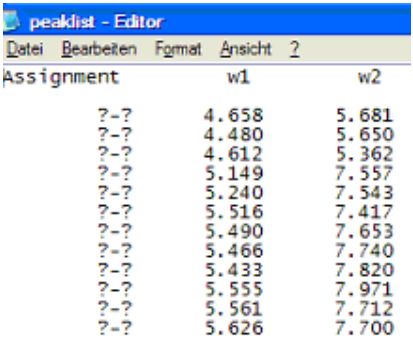


Abbildung 3.2: Peakliste

3.3 Distributions-Datei

Um im Programm eine prozentuale Zurodnung der Frequenzen auf eine Nukleinbase treffen zu können, wurden mittels der verschiedenen NMR-Experimente bewusst
Werte von bekannten Basen gemessen und bestimmte Informationen dieser Daten in eine Distributions-Datei geschrieben.

Eine solche Datei besitzt keine Kopfzeile und jede Zeile ist für ein Nukleinbasenpaar reserviert. Diese Zeilen beginnen immer mit einem 'X', um anzugeben, dass die gemessenen Werte der X-Achse folgen. Die einzelnen Informationen werden mit einem Leerzeichen voneinander abgegrenzt. Nun folgt das Basenpaar, für welches die Werte gemessen wurden und anschließend der Durchschnittswert sowie das Quadrat der Standardabwei-

chung und noch einmal die Standardabweichung ohne Abweichung. Ein 'Y' gibt an, dass die Werte der Y-Achse folgen. Diese Informationen unterscheiden sich nicht von denen der X-Werte, zusätzlich wird am Ende aber noch die Kovarianz angehängt.

Erkennen kann man diese Dateien an dem Anfangswort 'distrs', eine spezielle Dateiendung existiert nicht.

3.4 BRMB

Bei der BRMB handelt es sich um eine Datenbank, die viele Ergebnisse von NMR-Experimenten an Proteinen, Peptiden und Nukleinsäuren speichert. Sie wurde von der Universität von Wisconsin ins Leben gerufen und ist für das Programm vor allem nützlich um Fasta-Dateien herunter zu laden. Da in Zürich nur die Daten für die Distributions- sowie die Peak-Dateien durch Experimente gewonnen werden, benötigt man für das Programm noch die zugehörige Fasta-Datei. Diese kann Dank der integrierten Suchfunktion der Webseite (http://www.bmrb.wisc.edu/) schnell gefunden werden.

Kapitel 4

Konzept

4.1 Funktionsumfang

Bevor ein Konzept erstellt werden kann, muss festgelegt werden, welche Funktionen das Programm erfüllen soll. Dazu gehören die folgenden Dienste.

- Diagramm aus Peak-Liste erzeugen
- Möglichkeit bieten, 2D-Distributionen im Diagramm anzuzeigen
- Sekundärstruktur aus Fasta-File erzeugen
- Anzeige der Berechnung der Wahrscheinlichkeiten
- Druckfunktion für Berechnung, Sekundärstruktur und Diagramm bieten
- Hilfe bei der Auswertung von NMR-Experimenten

4.2 Designanforderungen

Die folgenden Punkte geben die wichtigsten Anforderungen an den Designer wieder. Sie stellen ein wichtiges Kriterium dar, um das Programm allen Interessenten zugänglich zu machen.

- Übersichtlichkeit bieten
- intuitive Bedienung
- zeiteffiziente Bearbeitung der Anfragen

4.3 Analyse und Konzeption

Die Frage nach der Programmiersprache konnte schnell beantwortet werden.

Da einige der Mitarbeiter Linux und andere Windows als Betriebssystem verwenden, bot sich Java an. Als Entwicklungsumgebung wurde Eclipse gewählt. Zu Beginn der Arbeit ein funktionierendes Konzept zu erstellen, ist allerdings ein sehr schwieriges Unterfangen, da auch während der Arbeit neue Aspekte und Änderungen hinzukommen. Drei wesentliche Anforderungen wurden an das Programm gestellt: die Auswertungen der Fasta- und Peakdateien, sowie die Darstellung als Sekundärstruktur und als Diagramm.

Bevor man mit dem Programmieren beginnen konnte, musste festgestellt werden, welche Datei-Informationen für welche Funktion gebraucht werden. Dabei werden die Sequenz-Daten aus der Fasta-Datei für die Darstellung der Sekundärstruktur sowie der Berechnung benötigt. Die Informationen aus der Peak-Datei werden für die Darstellung des Diagramms und zusätzlich auch noch für die Berechnung verwendet. Um auch später schnelle Änderungen vornehmen zu können, sollte für jedes Experiment eine eigene Methode zum Auslesen geschrieben werden, was bedeutet, das insgesamt vier Eingabefunktionen benötigt werden. Für den Benutzer sollen danach, entweder mit Betätigung eines extra Buttons oder ohne, das Diagramm, die Sekundärstruktur sowie die Auswertung sichtbar gemacht werden. Die Ausgabe des Diagrammes und der Sekundärstruktur sind im Gegensatz zu der Auswertung sehr komplexe Ausgabefunktionen, die jeweils mehrere eigene Klassen benötigen.

Die Auswertung hingegen benötigt nur einen zusammenhängenden Text auf einem scrollbaren Fenster.

Dennoch ist die Berechnung nicht trivial zu lösen und benötigt eine sehr komplexe Abfragefunktion mit den Informationen der Distributionsdateien. Zusätzlich sollen im Diagramm noch sogenannte 2D-Distributionen dargestellt werden. Diese geben die Informationen der Distributions-Datei graphisch im Diagramm an. Da es sich bei den 2D-Distributionen um eine sehr große Anzahl handelt, würde es sehr unübersichtlich werden, alle gleichzeitig darstellen zu wollen. Deswegen musste eine Alternative gefunden werden, die darin besteht, jede Distribution mit einem Knopfdruck sichtbar machen zu können. Dadurch wird allerdings der Bildschirm mit Buttons überschwemmt werden, weswegen die Idee geboren wurde, diese als ToolTip anzulegen. Sobald man mit der Maus über ein Label fährt, sollen die Buttons sichtbar werden.

Auf interne und externe Datenbestände muss zunächst nicht zugegriffen werden, weswegen dafür keine Methoden implementiert wurden.

Kapitel 5

Realisierung

Dieses Kapitel beschäftigt sich mit der Implementierung der einzelnen Funktionen.

5.1 Grafische Benutzeroberfläche

Zu Beginn der Arbeit musste eine Klasse geschrieben werden, die die wichtigsten Methoden der Benutzerschnittstelle sowie die Main-Methode enthält. Diese Klasse heißt „MainFrame“. Zunächst wird im Konstruktor ein neues Frame erstellt und mit Hilfe des Border-Layout's verschiedene Panels angeordnet. Das Border-Layout ordnet die Komponenten in allen Himmelsrichtungen an, so werden beispielsweise im Norden die Buttons zum Einlesen und Bearbeiten der Sequenz sowie einer Textarea für die eingelesene Sequenz mit je einem Bild links und rechts dargestellt (Abb. 5.1).

Für die grafischen Komponenten der Oberfläche wurde das Paket „swing“ verwendet, welches einerseits sehr moderne Komponenten liefert und andererseits kostenlos zur Verfügung gestellt wird. Damit der Benutzer auch eine Sequenz einlesen kann, die er bereits in den Zwischenspeicher kopiert hat, wurde eine Paste-Methode geschrieben, die bei einem Rechtsklick auf die Textarea aktiviert wird. Dies wurde mit der awt-Methode *textArea.setText(getToolkit().getSystemClipboard().getContents(this).getTransferData(DataFlavor.stringFlavor))* realisiert. Falls der Zwischenspeicher leer sein sollte, wird die daraufhin folgende Fehlermeldung aufgefangen und ein Hinweis darauf ausgegeben.

Eine Menüleiste wurde ebenfalls angelegt, um die Bedienung für alle, die sich an die Arbeit mit einer solchen Menüleiste gewöhnt haben, zu erleichtern. Dringend benötigt wird diese nicht, da fast alle Funktionen die sie bietet, auch auf der Oberfläche wieder zu finden sind. Außerdem wurden alle Punkte im Menü mit einer Schnell-Tasten-Kombination

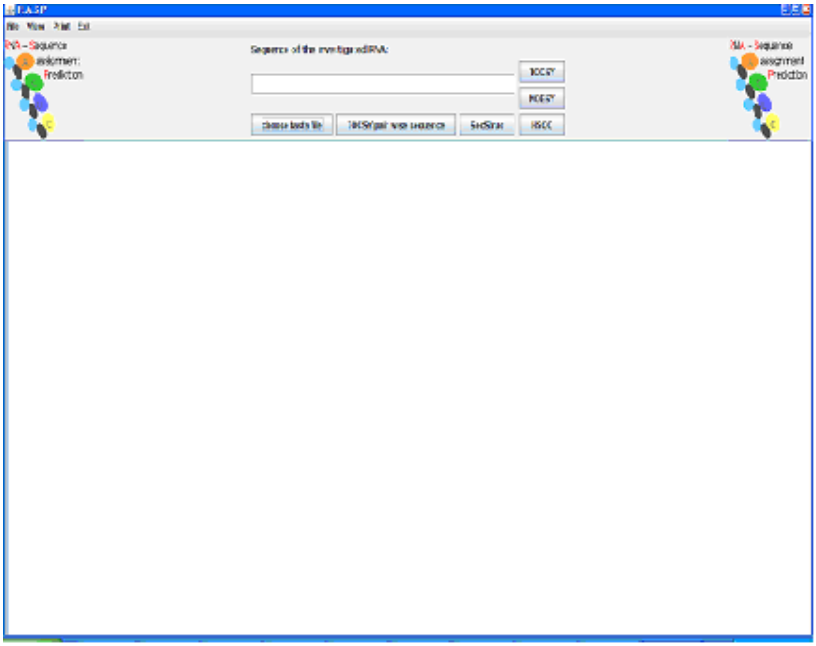


Abbildung 5.1: Startfenster des Programms RASP

versehen, um eine effektivere Arbeit zu ermöglichen. Sowohl in der Menüleiste wie auch im nördlichen Panel kann man die Buttons für die drei NMR-Experimente Tocsy, Noesy und HSQC wieder finden. Sobald man eins betätigt, wird man aufgefordert, eine Peak-Liste auszuwählen und einzulesen. Während des Vorgangs des Einlesens wird ein kleines Fenster mit der Aufforderung „Bitte warten“ angezeigt, damit es bei größeren Dateien nicht den Anschein erweckt, dass der Computer sich aufgehangen hat. Falls zu diesem Zeitpunkt noch keine Sequenz eingelesen wurde, wird eine Warnung angezeigt, dass keine Berechnung der Wahrscheinlichkeiten durchgeführt werden kann und es wird nur das Diagramm angezeigt. Hat man vorher bereits eine Sequenz eingelesen, so wird unter dem Diagramm zusätzlich noch ein ScrollPanel mit der Auswertung der beiden Dateien ausgegeben. Dabei werden zunächst die Koordinaten des Punktes angegeben, welcher aktuell verglichen wurde, um anschließend die drei wahrscheinlichsten Paarungen anzugeben. Dafür wird hinter der Paarung die Prozentanzeige dargestellt, um danach noch alle Paarungen aufzulisten, die diese Voraussetzung in der Sequenz erfüllen.

Für die Anzeige des Diagramms wurde JFreeChart importiert. Dieses frei zugängliche

Framework, bei dem nur die Dokumentation kostenpflichtig ist, hilft bei der Erstellung von verschiedenen Diagrammen. So können neben Säulen-, Torten-, Balken- und Gantt-diagrammen auch Histogramme erstellt und angezeigt werden. Um ein solches Diagramm anzeigen zu können, muss zunächst ein Datensatz angelegt werden. Dies geschieht in der Methode *createDataset()*, in der mit Hilfe einer foreach-Schleife alle Koordinaten aus der Peak-Datei in eine neu angelegte *XYSeries* geschrieben werden. Diese Serie wird verwendet, um ein neues Diagramm-Objekt anzulegen, dem man alle wichtigen Informationen, wie Name, Achsenbeschriftung, Serie, sowie die Information, ob eine Legende, url sowie ToolTips angezeigt werden sollen.

Nebenstehend ist der zugehörige Programmcode angegeben:
ChartFactory.createXYLineChart(title, xAxisLabel, yAxisLabel, dataset, orientation, legend, tooltips, urls).

Nachdem ein neuer *XYLineAndShapeRenderer()* angelegt wurde, kann mit dessen Hilfe festgelegt werden, wie die zu zeichnenden Punkte gestaltet und verbunden werden sollen. Es wird beispielsweise festgelegt, dass alle Punkte der Peak-Datei als Kreise mit dem Radius 4 gezeichnet werden (*renderer.setSeriesShape(0, new Ellipse2D.Float(0, 0, 4, 4))*. Weiterhin wird die Achsenbeschriftung invertiert, so dass der größte Wert im Koordinatenursprung liegt. Dabei handelt es sich um einen Standard bei der Darstellung einiger NMR-Experimente. Es wird auch festgelegt, dass sich die Achsenbeschriftung automatisch an den größten und kleinsten Punkt anpasst. Östlich des Diagramms werden mit Hilfe des BorderLayout's vier bis fünf Label's angelegt, die für je ein Nukleotid stehen. Diesen werden MouseMotionListener zugewiesen, die aktiviert werden, sobald man mit der Maus über ein Label fährt. Sobald dies geschieht, werden alle 2D-Distributionen angezeigt, die dem Label zugehörig sind. Dabei wird das erste Nukleotid der Paarung für den Vergleich verwendet.

Im Westen des Fensters kann, nachdem die TextArea eine Sequenz beinhaltet, auch das Fenster für die Sekundärstruktur aufgerufen werden. Falls die Sekundärstruktur aufgerufen wird, öffnen sich zunächst zwei Textfelder, in die man die Anzahl der im Stem und Loop vorhandenen Nukleotide angeben kann. Es genügt allerdings, wenn man nur ein Textfeld ausfüllt, das andere wird aus der Anzahl der Gesamtnukleotidenanzahl berechnet. Nun öffnen sich zwei Reiter, in denen die Sekundärstruktur der Sequenz dargestellt wird. In dem ersten Reiter kann man die Namen der einzelnen Nukleotide verändern, während im zweiten keine Veränderungen mehr möglich sind, aber statt dessen werden alle Purine (Adenin und Guanin) schwarz und alle Pyrimidine (Cytosin und Uracil) in Rot dargestellt. Angeordnet wurden diese mit Hilfe des GridBagLayout, wodurch man die einzelnen Punkte leichter im Fenster anordnen kann. Damit die Zuordnung der Nukleotide

aus der Sekundärstruktur zu den Punkten im Diagramm einfacher fällt, wurde auch ein ToolTip hinzugefügt, der sich öffnet, sobald man mit der Maus über das entsprechende Label fährt. Jedem ToolTip sind alle gefundenen Wahrscheinlichkeiten zugeordnet und sobald man eine der Wahrscheinlichkeiten auswählt, wird der dazu passende Punkt im Diagramm markiert, wodurch man leichter den Überblick behalten kann.

5.2 Algorithmen

> *Eine zur Lösung einer Klasse gleichartiger Probleme präzis formulierte Handlungsanweisung, die bei vorgegebenen Eingangsdaten eines festgelegten Typs mit Hilfe bekannter Regeln und unter Benutzung genau beschriebener Hilfsmittel unter Beachtung von Randbedingungen zu einem bestimmten Ergebnis führt, heißt Algorithmus.*
> Dietmar Henke

5.2.1 Algorithmus für die Darstellung der Sequenz

Um die Sequenz in verschiedenen Farben anzeigen zu können, wird bereits ein einfacher Algorithmus benötigt. Dieser beinhaltet eine einfache for-Schleife, die mit Hilfe der *length()*-Methode von Strings genau so lange zählt, bis das Ende der Sequenz erreicht ist. In jedem Durchlauf wird dabei überprüft, um welches Nukleotid es sich handelt, um es in der entsprechenden Farbe darzustellen sowie mit einer Nummer zu versehen. Da die Nukleotide fortlaufend ohne Unterbrechung nummeriert werden, wird die Variable der for-Schleife verwendet, die mit jedem Durchlauf inkrementiert wird. Um die Sequenz wieder in ihrer ursprünglichen Form darzustellen, existiert eine weitere for-Schleife, die mit einer if-Bedingung überprüft, an welcher Stelle Abkürzungen für Nukleotiden zu finden sind und diese einem String hinzufügt, der danach angezeigt wird.

5.2.2 Lese-Algorithmen

Für das Einlesen der Peak-Listen existieren weitere Algorithmen, die sich allerdings für alle drei Arten von Listen sehr ähnlich sind. Es handelt sich dabei um sogenannte foreach-Schleifen, die dafür bestimmt sind, ein Array oder einen Vektor solange abzuarbeiten, bis das Ende der Liste erreicht ist. In dem speziellen Fall lautet die Schleife *for(PeakData peakData : peakList)*. Die erste Variable (PeakData) gibt die Art des Objektes an, welches im Konstruktor in der Klasse *PeakData* festgelegt wurde. Es besteht aus einem String, der

den Namen der Peak-Datei beinhaltet, sowie zwei double-Variablen namens „w1“ und „w2“, die für die einzelne gemessene Frequenz stehen. Für jede Variable beinhaltet die Klasse noch jeweils eine get-Methode, die die gewünschte Information zurück gibt. Die zweite Variable gibt den Namen an und ist frei wählbar. Als Letztes folgt noch die Liste, aus der die Informationen herausgelesen werden sollen. Diese wird mit den Werten der vorher vom Benutzer ausgewählten Peak-Datei, mit Hilfe der Klassenmethode *public static List<PeakData> loadPeakDataFromFile(File inputFile)* der Klasse *DataHandler*, gefüttert. In dieser Methode wird zunächst die Datei mit einem BufferedReader (*BufferedReader in = new BufferedReader(new FileReader(inputFile))*) geöffnet, um sie in einer while-Schleife solange auszulesen, bis keine Daten mehr gefunden werden. Nun wird ein String-Array für alle erhaltenen Werte angelegt und mit dem Befehl *split(„(//s)+“)* gefüttert. Daraus ergeben sich die Daten für die PeakData, die anschließend auf ein neues Objekt zugewiesen werden. Alle Objekte werden in einer Liste gespeichert, die zurück geliefert wird. Diese kann beispielsweise mit der foreach-Schleife ausgelesen werden.

5.2.3 Algorithmus für die Kalkulation

Die Kalkulation gehört zu den aufwendigsten Algorithmen im Programm. Sie wird automatisch generiert, sobald alle erforderlichen Daten vorhanden sind. Zunächst wird wieder die Methode *public static List<PeakData> loadPeakDataFromFile(File inputFile)* der Klasse *DataHandler* aufgerufen, da die Daten für die Berechnung erforderlich sind. Die abgerufenen Informationen werden mit den drei Arten von Experimenten durch if-Anweisungen abgeglichen und die dazugehörige Distributions-Datei wird geöffnet. Diese Datei wird an die Methode *public static List<DistrsData> loadDistrsDataFromFile(String file_distrs)*, die sich auch in der Klasse *DataHandler* befindet, weitergeleitet. Der einzige Unterschied zu der Behandlung der Peak-Datei besteht darin, dass *DistrsData*-Objekte erschaffen und in einer Liste gespeichert werden. Ein solches Objekt besitzt die Informationen über den Basenpaarnamen, die Mittelwerte, Varianz und Standardabweichungen der x- und y-Koordinaten, sowie die Kovarianz. Anschließend werden beide Listen an die Methode *public String calculate(List<PeakData> peakList, List<DistrsData> distrsList)* der Klasse *Calculation* weitergegeben. Um die komplexen Berechnungen der Klasse verstehen zu können, muss man zunächst das Modell der multivariaten Normalverteilung kennen.

Das Modell besagt, sobald für verschiedene Datengruppen eine Wahrscheinlichkeitsdichtefunktion berechnet wird, kann für jedes unbekannte, in eine der Gruppen einzuordnende Wertepaar eine Wahrscheinlichkeit der Zugehörigkeit zu einer bestimmten Gruppe berechnet werden. Diese Wahrscheinlichkeit bestimmt sich aus dem Integral der Dichte-

funktion. Die allgemeine n-dimensionale Dichtefunktion *f(x)* lautet in der Matrixschreibweise:

$$f(x) = \frac{1}{\sqrt{2\pi^{\eta}}\sqrt{|K|}} exp\left\{-\frac{1}{2}\left[(\chi-\mu)^T K^{-1}(\chi-\mu)\right]\right\}. \tag{5.1}$$

Der Vektor x = $(x_1; x_2; \ldots; x_n)$ enthält hierbei die laufenden Variablen der Funktion. K ist die Kovarianz-Matrix und $\mu = (\mu_1; \mu_2; \ldots; \mu_n)$ der Mittelwertvektor, welcher die Mittelwerte aller Dimensionen bzw. Variablen enthält. Jeder Mittelwert μ_j mit *j* = 1, . . . , n ist definiert durch

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} X_{ji}. \tag{5.2}$$

X_{ji} ist hier der Messwert der Variablen *j* mit dem Index *i*. *m* ist die Anzahl der Messwerte. Die Kovarianzmatrix ist gegeben mit

$$K = \begin{pmatrix} Cov(x_1,x_1) & Cov(x_1,x_2) & \cdots & Cov(x_1,x_n) \\ Cov(x_2,x_1) & Cov(x_2,x_2) & \cdots & Cov(x_2,x_n) \\ \vdots & \vdots & \ddots & \vdots \\ Cov(x_n,x_1) & Cov(x_n,x_2) & \cdots & Cov(x_n,x_n) \end{pmatrix} \tag{5.3}$$

Die Stichprobenkovarianz $Cov(X_j, X_k)$ der beiden Variablen j und k erhält man über die Gleichung

$$Cov(X_j, X_k) = \frac{1}{m-1}\sum_{i=1}^{m}(X_{ji-\mu_j})(X_{ki-\mu_k}). \tag{5.4}$$

Alternativ kann sie auch durch Stichprobenvarianzen $Var(X_j)$ und $Var(X_k)$ und den Korrelationskoeffizienten ρ_{jk} ausgedrückt werden:

$$Cov(X_j, X_k) = \rho_{jk}\sqrt{Var(X_j}\sqrt{Var(X_k} = \rho_{jk}\sigma_j\sigma_k. \tag{5.5}$$

Die Stichprobenvarianz berechnet sich aus

$$Var(X_j) = \frac{1}{m-1} \sum_{i=1}^{m} (X_{ji} - \mu_j)^2 \tag{5.6}$$

und die Standardabweichung σ_j aus

$$\sigma_j = \sqrt{Var(X_j)}. \tag{5.7}$$

Dadurch lässt sich die Kovarianzmatrix auch über die Varianzen und Korrelationskoeffizienten ausdrücken:

$$K = \begin{pmatrix} \sigma_1^2 & \rho_{12}\sigma_1\sigma_2 & \cdots & \rho_{1n}\sigma_1\sigma_n \\ \rho_{21}\sigma_2\sigma_1 & \rho_{22}\sigma_2^2 & \cdots & \rho_{2n}\sigma_2\sigma_n \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{n1}\sigma_n\sigma_1 & \rho_{n2}\sigma_n\sigma_2 & \cdots & \sigma_n^2 \end{pmatrix} \tag{5.8}$$

Aus Gleichung 5.1, der n-dimensionalen Dichtefunktion, lässt sich zum Beispiel die Dichte der eindimensionalen Normalverteilung oder die für die Kalkulation notwendige Wahrscheinlichkeitsdichtefunktion der zweidimensionalen Normalverteilung $(n = 2, x = (x_1, x_2))$ sowohl unkorrelierter als auch korrelierter Daten (mit $\rho \neq 0$) herleiten.

Die eindimensionale Normalverteilung wird nicht weiter betrachtet, da sie für das Programm und damit diese Arbeit keine Rolle spielt. Angewendet auf die zweidimensionale Normalverteilung ergeben sich damit folgende Herleitungen:

$$K = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_2\sigma_1 & \sigma_2^2 \end{pmatrix}$$

$$|K| = (1 - \rho^2)(\sigma_1\sigma_2)^2$$

$$\sqrt{|K|} = \sigma_1\sigma_2\sqrt{1-\rho^2}$$

$$K^{-1} = \frac{1}{(1-\rho^2)} \begin{pmatrix} \frac{1}{\sigma_1^2} & -\frac{\rho}{\sigma_1\sigma_2} \\ -\frac{\rho}{\sigma_2\sigma_1} & \frac{1}{\sigma_2^2} \end{pmatrix}$$

$$\begin{pmatrix} x_1-\mu_1 \\ x_2-\mu_2 \end{pmatrix} K^{-1} \begin{pmatrix} x_1-\mu_1 \\ x_2-\mu_2 \end{pmatrix}^T = \frac{1}{1-\rho^2} x \left\{ \frac{(x_1-\mu_1)^2}{\sigma_1^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2}{\sigma_1\sigma 2} + \frac{(x_2-\mu_2)^2}{\mu_2^2} \right\}$$

$$f(x) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} exp \left\{ -\frac{1}{2(1-\rho^2)} \left[\frac{(x_1-\mu_1)^2}{\sigma_1^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2} \right] \right\} \tag{5.9}$$

Bei einer zweidimensionalen Normalverteilung ergibt sich das Volumen unter der Gaußglocke

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x_1, x_2) dx_1 dx_2 = 1. \tag{5.10}$$

Über das Integral der Gaußfunktion kann jedem Wertepaar $(X(_{ji}, X_{ki})$ eine Wahrscheinlichkeit zugeordnet werden. Da allen Punkten mit gleichem Funktionswert $f(x)$ die gleiche Wahrscheinlichkeit zugeordnet wird, ergibt sich bei korrelierten Daten eine Ellipse. Diese kann durch ihre quadratische Form definiert werden:

$$\tau = \left(\frac{\mu_1 - x_1}{\sigma_1} \right)^2 - \frac{2\rho(\mu_1 - x_1)(\mu_2 - x_2)}{\sigma_1\sigma_2} + \left(\frac{\mu_2 - x_2}{\sigma_2} \right)^2 \tag{5.11}$$

Dadurch wird allen Wertepaaren mit gleichem τ durch die Dichtefunktion der gleiche Funktionswert zugeordnet. Bei der Wahrscheinlichkeitsberechnung begrenzt die durch τ beschriebene Funktion das Integral der Dichtefunktion. Durch Einsetzen von Werten, die auf der Ellipse der einfachen Standardabweichung liegen (z.B. $\mu_1\mu_2 + \sigma_2$), erkennt

man, dass die elliptische Form der gemeinsamen einfachen Dichtefunktion $\tau = 1$ beträgt. Falls $\sigma_1 = \sigma_2$ und $\rho = 0$ sind, ergibt sich aus der ursprünglichen Ellipse ein Kreis. Der Zusammenhang zwischen r und der gemeinsamen t-fachen Standardabweichung ist quadratisch: $r = t^2$. Zur Volumenberechnung werden für die Variablen die Werte $\mu_1 = 0, \mu_2 = 0, \sigma_1 = 1, \sigma_2 = 1 und \rho = 0$ eingesetzt, wodurch sich folgende Dichtefunktion ergibt:

$$f_{SNV}(x_1, x_2) = \frac{1}{2\pi} exp\left\{-\frac{t^2}{2}\right\}$$

$$mitt^2 = r = x_1^2 + x_2^2 \tag{5.12}$$

Diese Gleichung ist rotationssymmetrisch um die x_3-Achse und gibt den Abstand des Wertepaares vom Koordinatenursprung an. Das Volumen unter ihr kann also anhand der Fläche vom Koordinatenursprung bis zur s-fachen Standardabweichung berechnet werden

$$V(s) = 2\pi \int_0^s t \frac{1}{2\pi} exp\left\{-\frac{t^2}{2}\right\} dt = \int_0^s t exp\left\{-\frac{t^2}{2}\right\} dt \tag{5.13}$$

Mit Hilfe der allgemeinen Substitutionsregel $\int f[g(t)]g'(t)dt = \int f(u)du$ (mit $u = g(t)$ und $du = g'(t)dt$) kann das Integral gelöst werden:

$$-\int f[g(t)]g'(t)dt = -\int -t exp\left\{-\frac{t^2}{2}\right\} dt$$

$$f[g(t)] = exp\left\{-\frac{t^2}{2}\right\}$$

$$u = g(t) = fract^2 2$$

$$g'(t) = -t$$

$$-\int f(u)du = -\int exp(u)du = -exp(u) = -exp - \frac{t^2}{2}$$

$$V(s) = \left[-exp\left\{-\frac{t^2}{2}\right\}\right]_0^s = 1 - frac1\sqrt{e^{s^2}}$$

$$s^2 = r$$

$$V(r) = 1 - frac1\sqrt{e^r} \tag{5.14}$$

Um die Wahrscheinlichkeit ermitteln zu können, mit der ein Wert zur Verteilung gehört, muss das Volumen unter der Glocke außerhalb der Grenze genommen werden:

$$f_{SNV}(x_1, x_2) = \frac{1}{2\pi} exp\left\{-\frac{t^2}{2}\right\}$$

$$P(X_1, X_2) = frac1\sqrt{e^r} \tag{5.15}$$

Das gesuchte Volumen unter der Dichtefunktion wird durch eine Ellipse begrenzt, die durch die quadratische Form r beschrieben wird. Es ist unabhängig von Form und Lage der Verteilung und wird allein durch den Wert von r bzw. t bestimmt. t ist der Wert für die $t - fache$ Standardabweichung. Die Volumenformel kann daher auf die Dichtefunktion der Ellipse übertragen werden. Der Anteil des Volumens am Gesamtvolumen unter der Fläche ist bei Kreis und Ellipse gleich. Die Wahrscheinlichkeitsformel für ein Wertepaar $(X1, X2)$ lautet somit:

$$P(X_1, X_2) = exp\left\{-\frac{1}{2}\left[\left(\frac{\mu_1 - X_1}{\sigma_1}\right)^2 - \frac{2\rho(\mu_1 - X_1)(\mu_2 - X_2)}{\sigma_1\sigma_2} + \left(\frac{\mu_2 - X_2}{\sigma_2}\right)^2\right]\right\} \tag{5.16}$$

Im Programm RASP wurde die Berechnung realisiert, indem zwei ineinander geschachtelte foreach-Schleife verwendet werden. Die erste Schleife durchläuft alle Daten der Peak-Datei und die zweite die gesamte Distributions-Datei. Innerhalb der beiden Schleifen wird die Formel mit den Werten der beiden Dateien angewendet. Die einzusetzenden Werte werden wie folgt zugeordnet:

- μ_1 = Mittelwert aller X-Werte der Distributions-Datei
- μ_2 = Mittelwert aller Y-Werte der Distributions-Datei
- X_1 = w2 der Peak-Datei
- X_2 = w1 der Peak-Datei
- σ_1 = Standardabweichung aller X-Werte der Distributionsdatei
- σ_2 = Standardabweichung aller Y-Werte der Distributionsdatei
- ρ = Kovarianz der Distributions-Datei

Damit erhält man für jeden Wert eine Wahrscheinlichkeit, die dann mit der Sequenz verglichen wird. Die drei höchsten Wahrscheinlichkeiten werden im Programm gespeichert, um sie in der Kalkulation wieder zu geben. Wie schon an der Formel zu erkennen ist, ergibt sich daraus eine exponentielle Komplexitätsfunktion. Um dies mit Werten aus dem Programm zu unterlegen, wurde eine Zeitmessung eingebaut, welche die Zeit in Millisekunden liefert. Da nur drei verschiedene Peaklisten und eine Fasta-Datei zur Verfügung standen, konnten nur für diese Messungen durchgeführt werden.

Experiment	Anzahl	Zeit
Tocsy	10	20ms
HSQC	67	260ms
Noesy	600	20000ms

Das Ergebnis zeigt, dass die Anzahl an eingelesenen Werten noch zu gering war, um eine schöne Exponentialfunktion zu verdeutlichen. Bei größeren Werten sollte die Zeit

aber immer schneller zunehmen, bis das Programm RASP irgendwann überfordert wäre. Für die Darstellung der 2D-Distributionen im Diagramm wird die gleiche Formel verwendet, nur dass hier die Werte der Peak-Datei mit einer for-Schleife simuliert werden. Diese durchläuft alle Werte zwischen Mittelwert - Standardabweichung bis Mittelwert + Standardabweichung. Damit erhält man eine Ellipse, die anzeigt, in welchem Bereich die größte Wahrscheinlichkeit für die entsprechenden Zuordnungen zu finden ist.

5.2.4 Algorithmus für die Anzeige der Sekundärstruktur

Ein weiterer Algorithmus ist in der Anordnung der Labels der Sekundärstruktur zu finden (Abb. 5.2). Im Stem werden diese paarweise und parallel zueinander angeordnet. Da hier ein *GridBagLayout* verwendet wird, brauchen nur die Werte der Y-Achse in einer for-Schleife hochgezählt werden. Die Anordnung im Loop ist wesentlich schwieriger, da die Anzahl genauso wie im Stem variabel sein kann und diese möglichst kreisförmig angeordnet werden müssen. Dafür wurde eine spezielle Methode geschrieben, die die Werte für die X-Achse berechnet und in einem Array speichert. Für die Anordnung wird nun für jeden einzelnen Wert überprüft, ob er gerade oder ungerade ist. Dies geschieht in einer Methode namens *protected boolean isOdd(int number).* Sobald die zu vergleichende Zahl und-verknüpft mit eins auch wieder eins ergibt, muss es sich um eine ungerade Zahl handeln, was sich die Methode zu nutze gemacht hat. Durch das Wissen, ob es sich um eine ungerade Zahl handelt, weiß man auch, auf welcher Seite des Loop's sie stehen muss, wodurch diese nur um eins nach außen geschoben werden muss, bis die Hälfte der aller im Loop vertretenen Label's erreicht ist, danach wird immer eins subtrahiert.

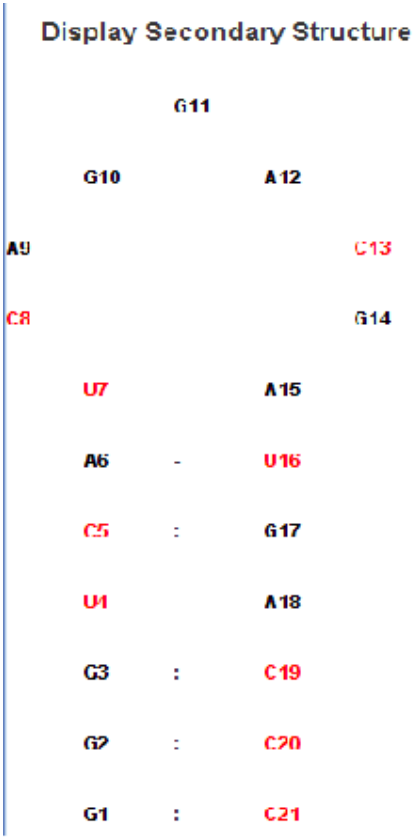


Abbildung 5.2: Sekundärstruktur

5.3 Anwendung auf eine Tocsy-Peakliste

Für ein besseres Verständnis der Arbeit wird hier die einfache Benutzung des Programmes RASP an einer Beispiel-Sequenz und einer Tocsy-Peakliste verdeutlicht. Um RASP starten zu können, benötigt man eine Java SE Runtime Umgebung (JRE) Version 6, welche man kostenlos unter dem Link „http://java.sun.com/javase/downloads/index.jsp“ herunterladen kann. Nachdem diese heruntergeladen und den Installations-Anweisungen gefolgt wurde, kann das Programm mit einem Doppelklick gestartet werden. Es öffnet sich ein Fenster, in dem neben der Menüleiste und dem Logo von RASP noch eine Textarea und sechs Knöpfe zu sehen sind (Abb. 5.1). Nun sollte man eine Sequenz auswählen, indem man entweder selbst eine in die Textarea einträgt oder den Knopf „choose fasta file“ betätigt, woraufhin sich ein *java.awt.FileDialog* öffnet (Abb. 5.3).

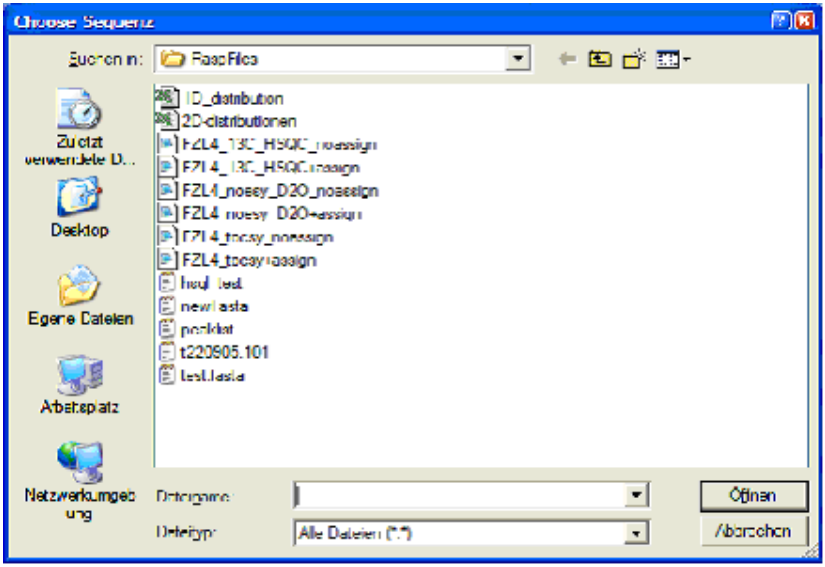


Abbildung 5.3: Filedialog des Programmes RASP

Nachdem eine Fasta-Datei ausgewählt wurde, erscheint die ausgelesene Sequenz in der Textarea. Für eine schönere Ansicht, in der die einzelnen Nukleotide nummeriert und Cytosin sowie Uracil rot hervorgehoben werden, kann man den Button namen's: „TOCSY-pair wise sequence“ betätigen (Abb. 5.4).

Es kann nun ein Experiment ausgewählt werden. Im abgebildeten Beispiel wird der Tocsy-

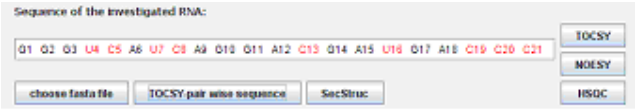


Abbildung 5.4: Nummerierte Darstellung der Sequenz

Knopf verwendet, woraufhin sich wieder ein *java.awt.FileDialog* öffnet, in dem man seine Peak-Datei auswählen kann. Sobald man sich für eine entschieden hat, öffnet sich das Diagramm und darunter die Kalkulation (Abb. 5.5).

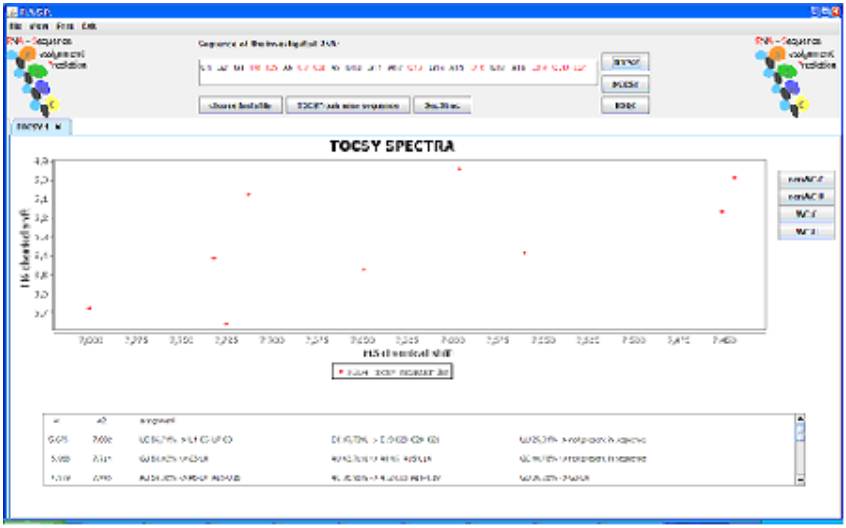


Abbildung 5.5: Diagramm und Kalkulation

Im Diagramm kann man die neun gemessenen Punkte des Experimentes erkennen. Sobald man mit der Maus über einen fährt, öffnet sich der ToolTip zu dem Punkt, in dem die Wahrscheinlichkeiten zu den Zuordnungen angegeben werden. An der rechten Seite sind die Buttons für die 2D-Distributionen zu finden. Falls man auf einen klickt, wird im Diagramm die Ellipse für die entsprechende Verteilung angezeigt. Für Abbildung 5.6 wurden alle Watson-Crick-Paarungen für das Nukleotid Cytosin ausgewählt.

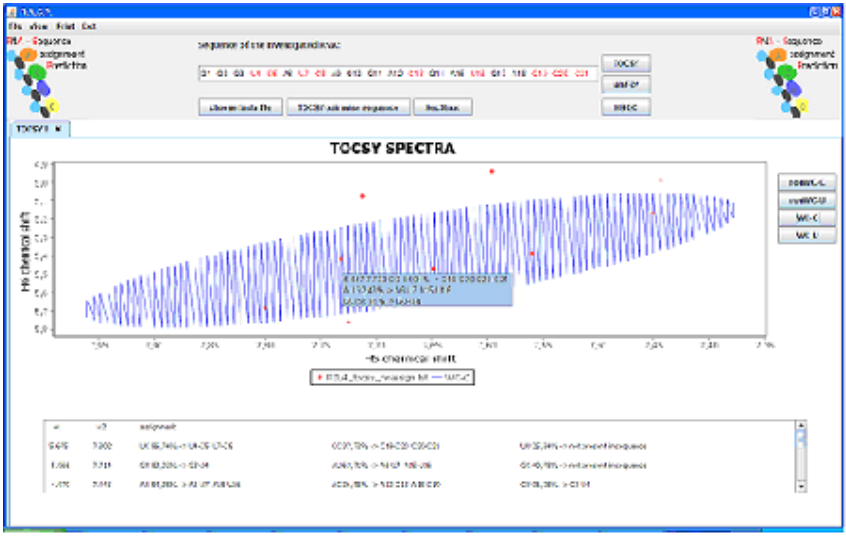
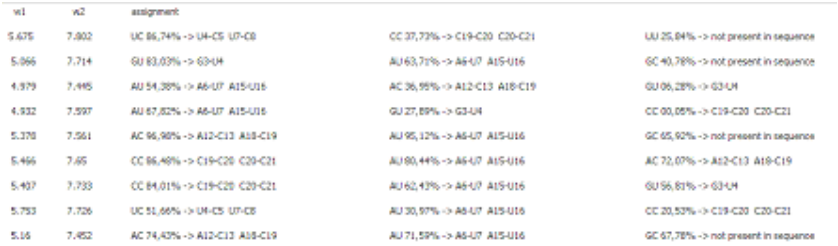


Abbildung 5.6: ToolTip und 2D-Distribution

Der ToolTip für einen Punkt innerhalb der Distribution zeigt mit einer sehr hohen Wahrscheinlichkeit, dass es sich hierbei um das Nukleotid Cytosin handelt, was auch die nahe Entfernung zum Zentrum der Distribution erklärt. Im Menü existiert noch eine weitere Funktion, mit der man die Kalkulation in einem eigenen Frame anzeigen kann. Dadurch wird dem Benutzer bei kleinen Peak-Dateien das Scrollen erspart (Abb. 5.7).



v1	v2	assignment		
5.675	7.802	UC 86,74% -> U4-C5 U7-C8	CC 37,73% -> C19-C20 C20-C21	UU 25,84% -> not present in sequence
5.066	7.714	GU 83,03% -> G3-U4	AU 63,71% -> A6-U7 A15-U16	GC 40,78% -> not present in sequence
4.979	7.445	AU 54,38% -> A6-U7 A15-U16	AC 36,95% -> A12-C13 A18-C19	GU 06,28% -> G3-U4
4.932	7.597	AU 67,82% -> A6-U7 A15-U16	GU 27,89% -> G3-U4	CC 00,05% -> C19-C20 C20-C21
5.378	7.561	AC 96,98% -> A12-C13 A18-C19	AU 95,12% -> A6-U7 A15-U16	GC 65,92% -> not present in sequence
5.466	7.65	CC 86,46% -> C19-C20 C20-C21	AU 80,44% -> A6-U7 A15-U16	AC 72,07% -> A12-C13 A18-C19
5.407	7.733	CC 84,01% -> C19-C20 C20-C21	AU 62,43% -> A6-U7 A15-U16	GU 56,81% -> G3-U4
5.753	7.726	UC 51,66% -> U4-C5 U7-C8	AU 30,97% -> A6-U7 A15-U16	CC 20,53% -> C19-C20 C20-C21
5.16	7.452	AC 74,43% -> A12-C13 A18-C19	AU 71,59% -> A6-U7 A15-U16	GC 67,78% -> not present in sequence

Abbildung 5.7: Kalkulation

Man kann in dem Fenster auch sehr gut die Koordinaten der Punkte erkennen und die drei größten Wahrscheinlichkeiten, die ihnen zugeordnet wurden. Der Text „not present in sequence“ gibt an, das die gefundene Zuordnung nicht in der Sequenz wieder gefunden wurde und deswegen unmöglich ist. Es wird außerdem immer eine Paarung angegeben, der die Wahrscheinlichkeit zugeordnet wird, nicht nur ein einzelnes Nukleotid. Falls man die Distributions-Datei verändert und beispielsweise nur die Informationen zu einem Wert anstatt einer Paarung einträgt, würde auch nur dieser eine Wert angezeigt werden. Das gestaltet allerdings das Auffinden in der Sequenz schwieriger, weswegen es nicht anzuraten ist.

Zum Abschluß besitzt man noch die Möglichkeit, den Button namens „SecStruc“ zu betätigen, wobei eine neue *javax.swing.JTabbedPane* mit zwei *javax.swing.JTextField*'s dargestellt wird. In dem einen kann man die Anzahl der Nukleotide im Stem und in dem anderen die im Loop angeben. Es reicht, eines der Felder auszufüllen, die Anzahl des anderen wird automatisch berechnet und eingetragen, da die Gesamtanzahl der Nukleotide in der Sequenz bekannt ist. Nach der Bestätigung öffnen sich zwei Reiter für die Anzeige der Sekundärstruktur. Im ersten Tab werden die Nukleotide in einem Textfeld dargestellt, um sie verändern zu können. Im zweiten Tab werden sie nur als *javax.swing.JLabel* und alle C- und U-Nukleotide rot dargestellt. Sobald man mit der Maus über ein solches Label fährt, wird der ToolTip geöffnet, in dem alle gefundenen Wahrscheinlichkeiten zu dem Nukleotid aus der Kalkulation angezeigt werden. Klickt man auf einen solchen Wert, wird der dazu gehörige Punkt im Diagramm farbig hervorgehoben (Abb. 5.8).

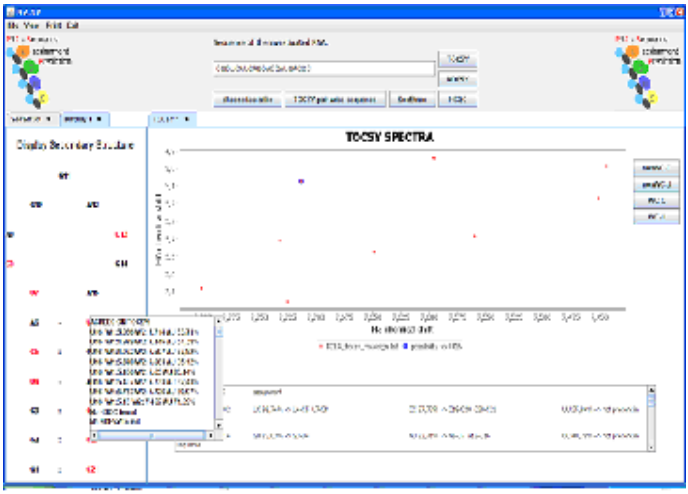


Abbildung 5.8: Diagramm mit einem hervorgehobenen Punkt aus der Sekundärstruktur

Kapitel 6

Fazit

6.1 Zusammenfassung

Durch die Entwicklung von RASP können Vorschläge für die Zuordnung von Messwerten zur Sequenz schneller unterbreitet werden. Weiterhin können die Daten visualisiert werden, wodurch es dem Benutzer leichter fällt, den Überblick bei größeren Datenmengen zu behalten. Zusammenfassend lässt sich sagen, dass fast alle geforderten Eigenschaften integriert werden konnten. Dazu gehört die Anzeige der Peak-Dateien als Diagramm, der Fasta-Datei als Sekundärstruktur und die Ausgabe der Wahrscheinlichkeiten. Es konnten sogar noch diverse Zusatzfunktionen eingebaut werden, wie das Umwandeln in eine pdf-Datei, ein zoombares Diagramm, die Möglichkeit zur Anzeige von 2D-Distributionen im Diagramm, ToolTip's im Diagramm sowie der Sekundärstruktur, die visuelle Zuordnung der Punkte in der Sekundärstruktur zum Diagramm und Tastaturkürzel für einen schnelleren Zugriff. Zusätzliche Wünsche, wie eine schönere Formatierung bei der Ausgabe der Kalkulation, konnten leider noch nicht eingebaut werden.

6.2 Ausblick

6.2.1 Tertiärstruktur

Nachdem die Primär- und Sekundärstruktur mit RASP dargestellt werden kann, soll auch die Möglichkeit zur Darstellung der Tertiärstruktur integriert werden. Dadurch bekommt man eine noch bessere visuelle Darstellung der eingelesenen Daten, ohne dass man auf Datenbanken wie der BRMB lange suchen muss. Das Java-Paket „jmol“ ist für eine solche Anzeige entwickelt wurden und könnte dafür beispielsweise verwendet werden.

6.2.2 Bindungen in Sekundärstruktur einbauen

Für eine noch deutlichere Darstellung der Sekundärstruktur sollen die einzelnen Verbindungen zwischen den Atomen dargestellt werden. Dadurch lässt sich besser verdeutlichen, welche Kräfte zwischen den Nukleotiden herrschen. Eine Vorstellung, wie das Ganze später aussehen wird, kann man aus Abbildung 6.1 entnehmen. Dabei soll jede Bindung, die im Bild durch Striche in verschiedenen Farben dargestellt werden, anklickbar sein und, sobald man diese aktiviert, sollen genauere Informationen zu der Verbindung ausgegeben werden. Die Anordnung der Atome um das Nukleotid wird vermutlich kreisförmig um das Molekül geschehen, da viele acht Atome besitzen und das Programm, wenn man diese wie auf dem Bild horizontal anordnet, zu unübersichtlich werden würde.

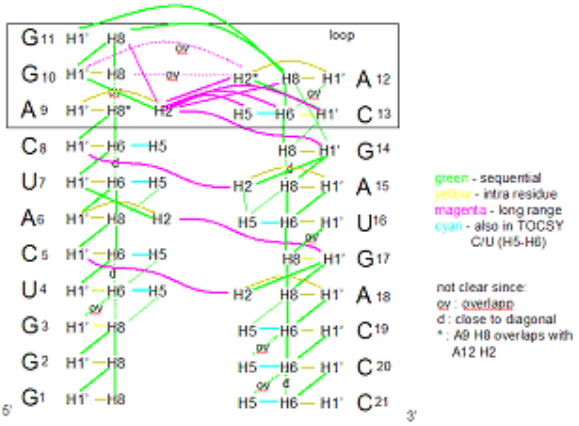


Abbildung 6.1: Veranschaulichung der Bindungen in der Sekundärstruktur (Urheberrecht: Mario Schubert)

6.2.3 Datamining

Sobald alle benötigten Funktionen in RASP fehlerfrei laufen, wäre auch Datamining aus bekannten Datenbanken eine Möglichkeit zur Erweiterung. Durch die Vielzahl an Messwerten könnten diese in einzelne Gruppen unterteilt werden, wodurch beispielsweise untersucht werden kann, wie sich Fehlerpaarungen auf chemische Verschiebungen auswirken. Die Daten können zusätzlich zum ersten auch noch zum zweiten Vorgänger separiert werden, wodurch man diese genauer in die Sequenz einordnen kann.

6.2.4 Schönheitsfehler und Erweiterung der Experimente

Auch kleinere Schönheitsoperationen würden RASP sehr gut tun, wie beispielsweise eine schönere Formatierung bei der Ausgabe der Berechnung der Wahrscheinlichkeiten oder die Möglichkeit, nach der Anzeige des Diagramms in einem eigenen Frame, das Diagramm wieder zurück in das Anfangs-Fenster zu integrieren. Es existieren mit hoher Wahrscheinlichkeit auch noch einige, bisher nicht gefundene Fehler, weswegen es sinnvoll wäre, das Programm einigen Testläufen zum Finden von Fehlern zu unterziehen. Es können zusätzlich zu den Tocsy-, HSQC- und Noesy-Experimenten auch noch weitere NMR-Experimente in das Programm integriert werden. Dazu könnten beispielsweise die Experimente HETCOR (Heteronuclear correlation), COSY (Correlated Spectroscopy), HMQC (Heteronuclear multiple quantum coherence), ROESY (Rotating frame Overhauser Enhancement Spectroscopy) oder EXSY (Exchange Spectroscopy) gehören.

Selbstständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Mittweida, den 29. Oktober 2009

_____

Sebastian Eube

Anhang A

Literaturverzeichnis

Fribolin, Horst: Ein -und zweidimensionale NMR-Spektroskopie
WILEY-VCH Verlag, 4.Auflage, 2006.

Cadenhead, Rogers; Lemay, Laura: Java 2 in 21 Tagen
Markt + Technik Verlag, 2003.

Alberts, Johnson, Lewis, Raff, Roberts, Walter: Molekularbiologie der Zelle
WILEY-VCH Verlag, 2.Auflage, 1995.

Samaschke, Karsten: Java 6, Einstieg für Anspruchsvolle
Addison-Wesley-Verlag, 2007

Informationen zu Francis Craic: http://www.whoswho.de/templ/te_bio.php?PID=474&RID=1 am 20.09.2009
Definition Algorithmus: http://www.henked.de/begriffe/algorithmus.htm am 20.09.2009

Anhang B

Tätigkeitsbericht in Stichpunktform

- Einlesen in das Thema
- Einführung in die Arbeit mit den Datenbanken BMRB und PDB
- Programmierung eines Frame mit einer Menüleiste
- Einfügen des Tocsy- und Fasta-File-FileChooser
- Programmierung der Kalkulation inklusive Berechnung
- Programmierung eines statischen Diagramms für TOCSY
- Programmierung Druckfunktionen
- Neu-Programmierung des Diagramms mit Hilfe des JFreeChart-Paketes
- Behebung eines Fehlers, der auftrat, falls man nicht den „view sequence“ Button vor dem einlesen der Peak-Liste betätigt hat
- Close-Button für die Sequenz eingefügt
- Tastaturkürzel für alle Anwendungen hinzugefügt
- Ersten ToolTip integriert
- Programmierung Sekundärstruktur inklusive fast aller Berechnungen und zusätzlichen Anzeigen mit dem null-Layout
- Alle Frames als Tab‘s zusammen fügen, um für eine bessere Übersichtlichkeit zu sorgen
- Anordnung der einzelnen Panels auf dem Frame
- 2D-Distribution für Tocsy-eingefügt
- File-Chooser für HSQC und Noesy eingefügt
- Programmierung HSQC und NOESY-Diagramme

- Verbesserten ToolTip für alle Diagramme hinzugefügt
- Anzeige der Sekundärstruktur auf GridBagLayout umgeschrieben
- Alle File-Chooser verändert, damit sie moderner aussehen
- Namen einiger Buttons verändert sowie Informations-Labels integriert
- Bei „view Sequenz“-Button bei zweimaliger Benutzung Ursprungszustand herstellen
- Menüpunkte auch als Knöpfe im nördlichen Panel zugänglich machen
- Informationen zur Peak-Liste im Diagramm integriert
- Möglichkeit mit Hilfe von iText die Kalkulation in ein pdf-File umzuwandeln
- Möglichkeit zum Schließen der Tabs integriert
- „Zuletzt geöffnetes Fenster anzeigen“ eingefügt
- Automatische Berechnung der im Stem-und Loop enthaltenen Informationen hinzugefügt
- Tabs mit Nummern versehen
- Kalkulation unter dem Diagramm angeordnet und es wird automatisch mit berechnet sobald das Diagramm aufgerufen wird
- „bitte warten“-Anzeige während den Berechnungen integriert
- Einige kleinere Fehler behoben, so dass die Reihenfolge, in der man die Knöpfe betätigt, keine Rolle mehr spielt
- Fehler behoben, so dass die Informationen der Sekundärstruktur wieder verändert werden können
- 2D-Distributionen auch bei HSQC und Noesy hinzugefügt
- Möglichkeit „angezeigte 2D-Distributionen wieder zu entfernen“ integriert
- Knöpfe für die 2D-Distribtionen ähnlich wie ToolTips anzeigen lassen, um Platz zu sparen
- In der Sekundärstruktur ToolTip hinzugefügt
- Sobald man auf eine Wahrscheinlichkeit der in der Sekundärstruktur dargestellten ToolTips drückt, wird der entsprechende Punkt im Diagramm hervorgehoben
- Alle weiteren bekannten Fehler behoben

Anhang C

Klassendiagramm

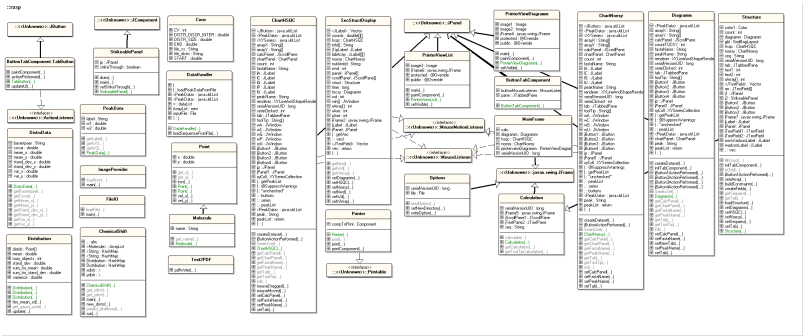


Abbildung .1: gesamtes Klassendiagramm

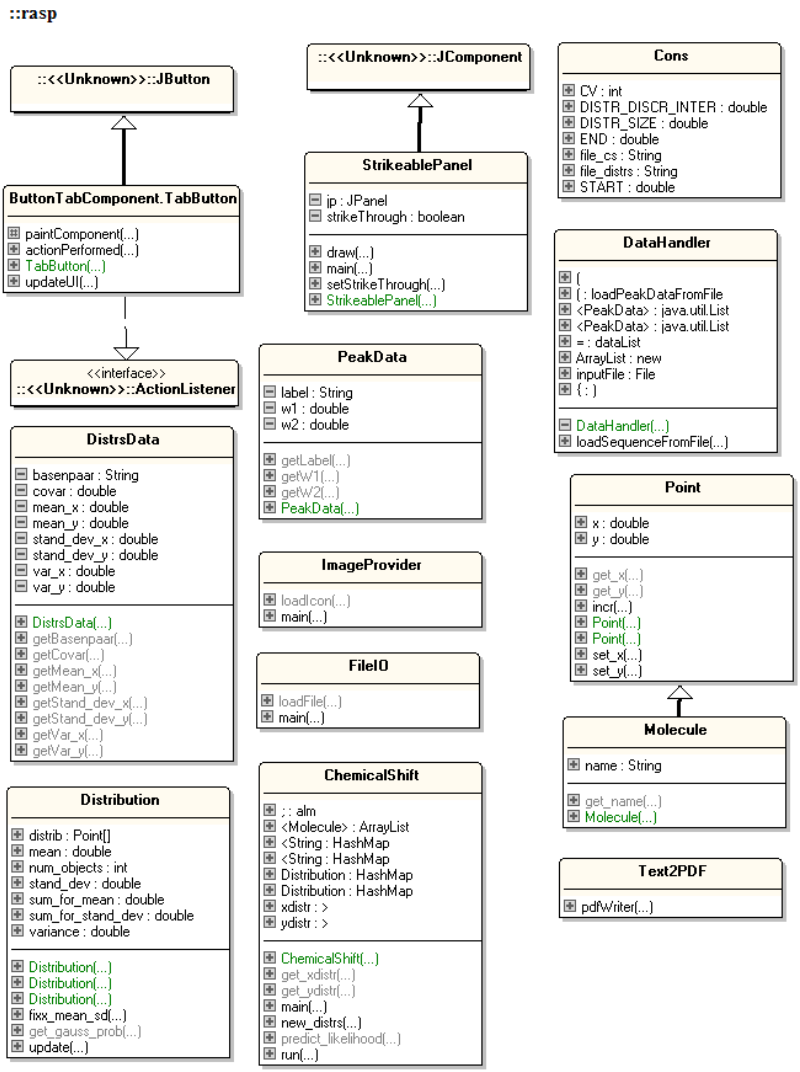


Abbildung .2: Teil 1 des Klassendiagramms

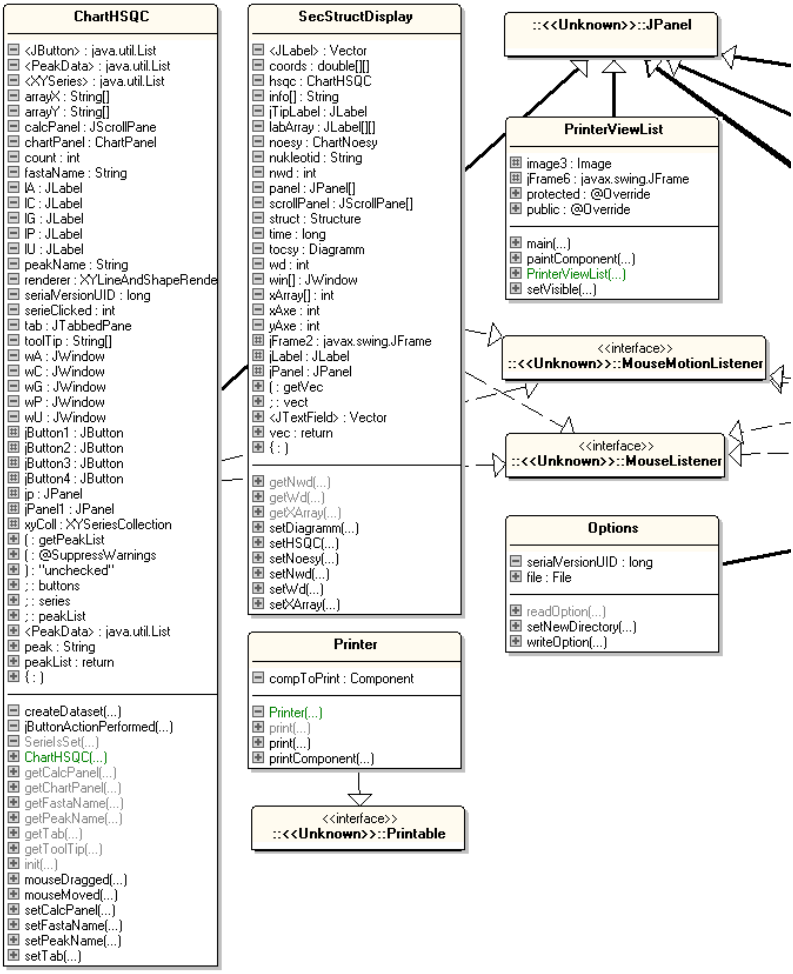


Abbildung .3: Teil 2 des Klassendiagramms

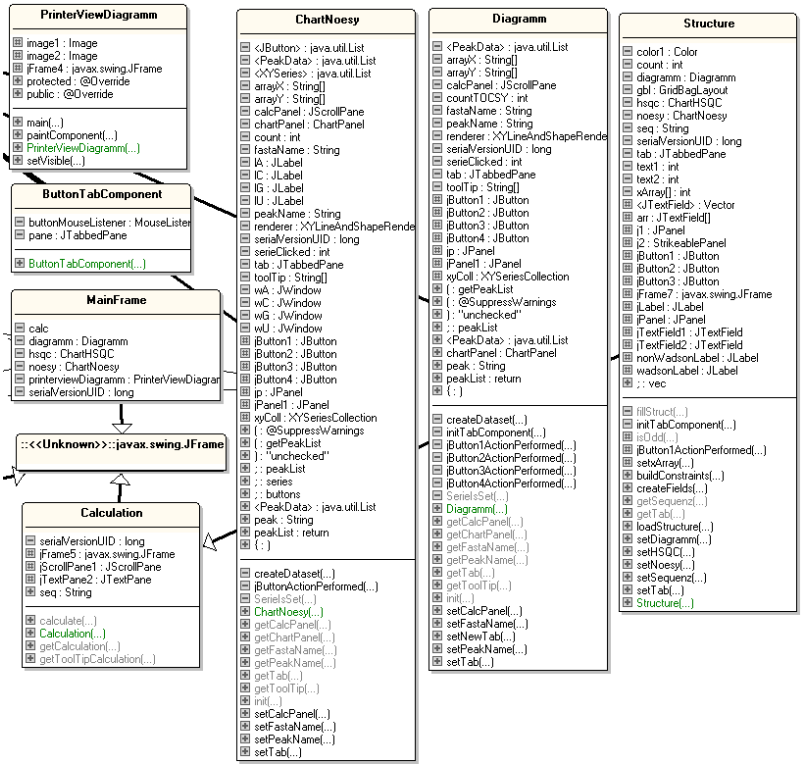


Abbildung .4: Teil 3 des Klassendiagramms

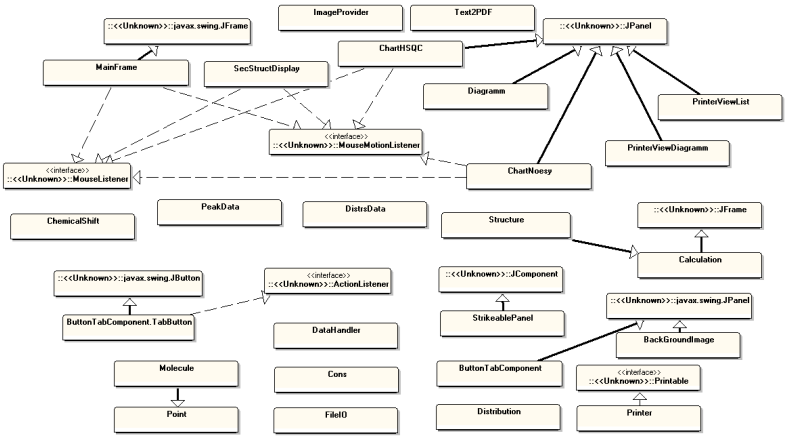


Abbildung .5: Klassendiagramm ohne Methoden

Calculation.java

```
1
2 package rasp;
3
4 import java.awt.Color;
13
14 /**
15  * @author SEUBE
16  */
17 public class Calculation extends JFrame {
18     /**
19      *
20      */
21     private static final long serialVersionUID = 1L;
22     protected JFrame jFrame5;
23     protected JTextPane jTextPane2;
24     protected JScrollPane jScrollPane1;
25 //    private java.util.List<DistrsData> distrsList;
26 //    private java.util.List<PeakData> peakList;
27     public String seq = null;
28
29   public Calculation(){
30     jFrame5 = new JFrame("List");
31     jScrollPane1 = new JScrollPane();
32     jTextPane2 = new JTextPane();
33     jFrame5.setBackground(Color.WHITE);
34     jFrame5.setBounds(new java.awt.Rectangle(585, 350, 450, 440));
35     jFrame5.add(jScrollPane1);
36     jScrollPane1.add(jTextPane2);
37     jScrollPane1.setAutoscrolls(true);
38     jScrollPane1.setHorizontalScrollBar(null);
39     jTextPane2.setBackground(Color.WHITE);
40     jTextPane2.setFont(new Font("Tahoma", Font.PLAIN, 11));
41     jTextPane2.setEditable(false);
42     jScrollPane1.setViewportView(jTextPane2);
43
44   }
45
46   public String calculate(List<PeakData> peakList, List<DistrsData> distrsList,
   boolean marker){
47         long start = new Date().getTime();
48         if(peakList != null && peakList.isEmpty() == false) {
49             StringBuilder builder = new StringBuilder();
50             builder.append("    w1" + "\t  w2" + "\tassignment\n\n");
51
52             for(PeakData peakData : peakList) {
53
54               double w1 = peakData.getW1();
55               double w2 = peakData.getW2();
56               double a, b, c, t = 0.0;
57               double likelihood = 0.0;
58               double l1 = 0.0;
59               double l2 = 0.0;
60               double l3 = 0.0;
61               String bp1 = null;
62               String bp2 = null;
63               String bp3 = null;
64               String impossible = "impossible";
65               StringBuilder d = new StringBuilder();
66               StringBuilder e = new StringBuilder();
67               StringBuilder f = new StringBuilder();
68               int countOdd = 0;
69 //              System.out.println("w1: " + w1 + "\tw2: " + w2);
70               for(DistrsData distrsData : distrsList){
71                   String basenpaar = distrsData.getBasenpaar();
```

```
72                  double mean_x = distrsData.getMean_y();
73                  double mean_y = distrsData.getMean_x();
74                  double stand_dev_x = distrsData.getStand_dev_y();
75                  double stand_dev_y = distrsData.getStand_dev_x();
76                  double covar = distrsData.getCovar();
77 //               System.out.println("Basenpaar: " + basenpaar + "\t" +
78 //                       "\tMean_x: " + mean_x + "\tMean_y:" + mean_y +
79 //                       /*"\tVar_x:" + var_x + "\tVar_y:" + var_y +*/
80 //                       "\tStand_dev_x:" + stand_dev_x + "\tStand_dev_y:" +
   stand_dev_y +
81 //                       "\tcovar:" + covar);
82 //               System.out.println("-------------");
83
84                a = Math.pow(((mean_x - w2)/stand_dev_x), 2.0);
85                b = 2 * covar * (mean_x - w2)*(mean_y - w1)/(stand_dev_x *
   stand_dev_y);
86                c = Math.pow(((mean_y - w1)/stand_dev_y), 2.0);
87                t = (a - b + c);
88
89                likelihood = (1 / (Math.sqrt(Math.pow(Math.E, t))));
90                if(likelihood==0.0){
91                    mean_x = distrsData.getMean_x();
92                    mean_y = distrsData.getMean_y();
93                    stand_dev_x = distrsData.getStand_dev_x();
94                    stand_dev_y = distrsData.getStand_dev_y();
95                    a = Math.pow(((mean_x - w2)/stand_dev_x), 2.0);
96                    b = 2 * covar * (mean_x - w2)*(mean_y - w1)/(stand_dev_x *
   stand_dev_y);
97                    c = Math.pow(((mean_y - w1)/stand_dev_y), 2.0);
98                    t = (a - b + c);
99
100                   likelihood = (1 / (Math.sqrt(Math.pow(Math.E, t))));
101               }
102 //              System.out.println("a: " + a + "\tb: " +  b + "\tc: " + c + "\tt:
   "+ t);
103 //              System.out.println("Likelihood " + basenpaar + ": " + likelihood);
104 //              System.out.println("l: "+likelihood);
105             if(likelihood > l1){
106                 l2 = l1;
107                 l1 = likelihood;
108                 bp2 = bp1;
109                 bp1 = basenpaar;
110             }
111             else{
112              if(likelihood > l2){
113                     l3 = l2;
114                     l2 = likelihood;
115                     bp3 = bp2;
116                     bp2 = basenpaar;
117                 }
118                 else{
119                    if(likelihood > l3){
120                    l3 = likelihood;
121                    bp3 = basenpaar;
122                    }
123                  }
124                }
125             }
126 //             System.out.println(seq);
127 //             System.out.println("seq: "+seq);
128              int laenge = seq.length();
129              String[] tokens = seq.split("");
130              String[] compareString = new String[4];
131              for (int i = 1; i < laenge; i++){
```

```
132                 tokens[i] = tokens[i] + tokens[i+1];
133
134                 if((bp1!=null)&&(bp2!=null)&&(bp3!=null)){
135                     if((bp1.charAt(0)=='P')||(bp2.charAt(0)=='P')||(bp3.charAt(0)
    =='P')){
136                         tokens[i]= tokens[i]+tokens[i+1];
137 //                        compareString = getCompareString(bp1, tokens[i]);
138                         if( isCompareString(bp1, tokens[i])&& l1 > 0.0001){
139                             int num = i;
140                             String[] dsplit = tokens[i].split("");
141                             String d1 = dsplit[1] + num + "-" + dsplit[2] + (num+1)
    + "-"+dsplit[3]+ (num+2) +"  ";
142                             d.append(d1);
143                         }else{
144                             if(l1<= 0.0001){
145                                 d.replace(0, laenge, impossible);
146                             }
147                         }
148
149                         if(isCompareString(bp2, tokens[i]) && l2 > 0.0001){
150                             int num = i;
151                             String[] esplit = tokens[i].split("");
152                             String e1 = esplit[1] + num + "-" + esplit[2] + (num+1)
    + "-"+esplit[3]+ (num+2) +"  ";
153                             e.append(e1);
154                         }else{
155                             if(l2<= 0.0001){
156                                 e.replace(0, laenge, impossible);
157                             }
158                         }
159
160                         if(isCompareString(bp3, tokens[i]) && l3 > 0.0001){
161                             int num = i;
162                             String[] fsplit = tokens[i].split("");
163                             String f1 = fsplit[1] + num + "-" + fsplit[2] + (num+1)
    + "-"+fsplit[3]+ (num+2) +"  ";
164                             f.append(f1);
165                         }else{
166                             if(l3<= 0.0001){
167                                 f.replace(0, laenge, impossible);
168                             }
169                         }
170
171                     }else{
172
173                         if(tokens[i].equals((Character.toString(bp1.charAt(0))
    +Character.toString(bp1.charAt(1)))) && l1 > 0.0001){
174                             int num = i;
175                             String[] dsplit = tokens[i].split("");
176                             String d1 = dsplit[1] + num + "-" + dsplit[2] + (num+1)
    + "  ";
177                             d.append(d1);
178                         }else{
179                             if(l1<= 0.0001){
180                                 d.replace(0, laenge, impossible);
181                             }
182                         }
183
184                         if(tokens[i].equals((Character.toString(bp2.charAt(0))
    +Character.toString(bp2.charAt(1)))) && l2 > 0.0001){
185                             int num = i;
186                             String[] esplit = tokens[i].split("");
187                             String e1 = esplit[1] + num + "-" + esplit[2] + (num+1)
    + "  ";
```

```
188                             e.append(e1);
189                         }else{
190                             if(l2<= 0.0001){
191                                 e.replace(0, laenge, impossible);
192                             }
193                         }
194 
195                         if(tokens[i].equals((Character.toString(bp3.charAt(0))
    +Character.toString(bp3.charAt(1)))) && l3 > 0.0001){
196                             int num = i;
197                             String[] fsplit = tokens[i].split("");
198                             String f1 = fsplit[1] + num + "-" + fsplit[2] + (num+1)
    + "  ";
199                             f.append(f1);
200                         }else{
201                             if(l3<= 0.0001){
202                                 f.replace(0, laenge, impossible);
203                             }
204                         }
205                       }
206                     //System.out.println(laenge + "\t" + i);
207                     //System.out.println(tokens[i]);
208                 }
209                 }
210 
211 //                if(bp1.charAt(4)!='i'){
212 //                  bp1 = bp1.charAt(0)+"<u>"+bp1.charAt(1)+"</u>"+bp1.substring(2);
213 //                }else{
214 //                  bp1 = "<u>"+bp1.charAt(0)+"</u>"+bp1.charAt(1)+bp1.substring(2);
215 //                }
216                 if(d.length() == 0){d.append("not present in sequence");}
217                 if(e.length() == 0){e.append("not present in sequence");}
218                 if(f.length() == 0){f.append("not present in sequence");}
219                 if(d.length() < 11){d.append("\t");}
220                 if(e.length() < 11){e.append("\t");}
221                 if(f.length() < 11){f.append("\t");}
222                 DecimalFormat df = new DecimalFormat("00.00");
223 //
    if((d.toString().contains(impossible))&&(e.toString().contains(impossible))&&(f.toS
    tring().contains((impossible))))
224 //                  builder.append("  " + w1 + " \t" + w2 + "
    \t"+impossible+"\t\n\t\t\n");
225                 if(!marker)
226                     builder.append("  " + w1 + " \t" + w2 + " \t" +
227                             bp1 + " " + df.format((l1 * 100)) + "%" + " -> " + d +
    "\n\t\t " +
228                             bp2 + " " + df.format((l2 * 100)) + "%" + " -> " + e +
    "\n\t\t " +
229                             bp3 + " " + df.format((l3 * 100)) + "%" + " -> " + f +
    "\n\t\t" + "\n ");
230                 else{
231 
232                     builder.append("  " + w1 + " \t" + w2 + " \t" +
233                             bp1 + " " + df.format((l1 * 100)) + "%" + " -> " + d +
    "\t\t " +
234                             bp2 + " " + df.format((l2 * 100)) + "%" + " -> " + e +
    "\t\t " +
235                             bp3 + " " + df.format((l3 * 100)) + "%" + " -> " + f +
    "\t\t" + "\n\n ");
236 
237                     countOdd++;
238                 }
239             jTextPane2.setText(builder.toString());
240 
```

```
241             }
242             System.out.println("Zeit: "+(new Date().getTime() - start));
243 //            builder.append("</html>");
244             jTextPane2.setText(builder.toString());
245             return builder.toString();
246 
247         }return null;
248     }
249 
250 //    @Override
251     public JScrollPane getCalculation() {
252         return jScrollPane1;
253 //        jFrame5.setVisible(b);
254     }
255 
256     public String[] getToolTipCalculation(List<PeakData> peakList, List<DistrsData>
    distrsList){
257         String calculation = calculate(peakList, distrsList, false);
258         String[] array = calculation.split("\n");
259         String[] tipArray = new String[array.length/4];
260         int k = 0;
261         for (int i = 2; i < array.length-4; i=i+4) {
262             tipArray[k] = "<html>"+array[i]+"<br>\n"+array[i+1]+"<br>\n"+array[i+2]
    +"</html>";
263             k++;
264     }
265 
266     return tipArray;
267     }
268     protected boolean isOdd(int number) {
269         number = number & 1;
270         if(number==1)
271             return false;
272         else
273             return true;
274     }
275     private boolean isCompareString(String basepair, String s){
276         String compareArray[] = new String[4];
277         if((basepair.charAt(1)=='u')&&(basepair.charAt(4)=='u')){
278             compareArray[0] = "GG";
279             compareArray[1] = "GA";
280             compareArray[2] = "AG";
281             compareArray[3] = "AA";
282         }
283         if((basepair.charAt(1)=='y')&&(basepair.charAt(4)=='u')){
284             compareArray[0] = "CG";
285             compareArray[1] = "CA";
286             compareArray[2] = "UG";
287             compareArray[3] = "UA";
288         }
289         if((basepair.charAt(1)=='u')&&(basepair.charAt(4)=='y')){
290             compareArray[0] = "GC";
291             compareArray[1] = "GU";
292             compareArray[2] = "AC";
293             compareArray[3] = "AU";
294         }
295         if((basepair.charAt(1)=='y')&&(basepair.charAt(4)=='y')){
296             compareArray[0] = "CC";
297             compareArray[1] = "CU";
298             compareArray[2] = "UC";
299             compareArray[3] = "UU";
300         }
301         for(int i=0; i<4; i++){
302             if(s.equals((Character.toString(compareArray[i].charAt(0))
```

```
   +Character.toString(basepair.charAt(2))
   +Character.toString(compareArray[i].charAt(1)))))
                    return true;
        }
        return false;
    }
    public static void main(String[] args) {
        Calculation calc = new Calculation();
        calc.setVisible(true);
    }

}
```

ANHANG E

Quellcode der Klasse zur Darstellung des Noesy-Experimentes

ChartNoesy.java

```
1 package rasp;
2
3 import java.awt.*;
33
34 /**
35  * @author SEUBE
36  */
37 public class ChartNoesy extends JPanel implements MouseListener,
    MouseMotionListener{
38
39 /**
40      *
41      */
42     private static final long serialVersionUID = 1L;
43 //    protected JFrame jFrame3;
44     private java.util.List<PeakData> peakList;
45 //    private final Color YELLOW2 = new Color(255, 255, 0, 50);
46 //    private final Color BLUE2 = new Color(0, 102, 255, 50);
47 //    private final Color RED2 = new Color(255, 51, 51, 50);
48 //    private final Color GREEN2 = new Color(102, 155, 102, 50);
49     protected JButton jButton1;
50     protected JButton jButton2;
51     protected JButton jButton3;
52     protected JButton jButton4;
53
54     protected JPanel jPanel1;
55     protected JPanel jp;
56     protected XYSeriesCollection xyColl;
57
58     String peak = null;
59     private JTabbedPane tab;
60     private String peakName;
61     private String[] arrayY;
62     private String[] arrayX;
63     private String[] toolTip;
64     private String fastaName;
65     private JScrollPane calcPanel;
66     private ChartPanel chartPanel;
67     private int count = 0;
68     private List<XYSeries> series;
69     private List<JButton> buttons;
70     private JWindow wA;
71     private JWindow wU;
72     private JWindow wG;
73     private JWindow wC;
74     private JLabel lU;
75     private JLabel lG;
76     private JLabel lC;
77     private JLabel lA;
78     private int serieClicked = 0;
79     private XYLineAndShapeRenderer renderer;
80
81     /**
82      * @return the calcPanel
83      */
84     public JScrollPane getCalcPanel() {
85         return calcPanel;
86     }
87     /**
88      * @param calcPanel the calcPanel to set
89      */
90     public void setCalcPanel(JScrollPane c) {
91         calcPanel = new JScrollPane();
92         calcPanel = c;
```